[]pgf@stop
.|[]pgf@stop

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Využití prostředků Scilab pro simulace zpracování signálů

Scilab System in DSP

2016 Kristýna Páleníková

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Kristýna Páleníková**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2601R013 Telekomunikační technika

Téma: Využití prostředků Scilab pro simulace zpracování signálů
Scilab System in DSP

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit podrobný návod pro práci se systémem Scilab jako alternativou komerčního systému MATLAB a Simulink při zpracování signálů.

1. Popište instalaci systému Scilab (Linux, Windows). Okomentujte jednotlivé typy instalace (Full, Custom, Minimal).
2. Layout systému a význam jednotlivých prvků. Možnosti konfigurace systému.
3. Popište práci v Consoli, v editoru SciNotes včetně případných možností debuggingu (možnosti psaní komentářů), práci s dokumentací (Help).
4. Popište tvorbu a spouštění skriptů a funkcí.
5. Popište možnosti práce se 2D a 3D grafů a práci s osami v rámci okna "Figure".
6. Popište možnosti importu modulů z Command Line a s pomocí správce ATOMS.
7. Popište práci s externími soubory - ukládání, nahrávání a zápis do souboru.
8. Popište možnosti efektivní výměny dat mezi systémy MATLAB a Scilab.
9. Popište možnosti práce se vstupně/výstupními zařízeními (např. zvuková karta).
10. Popište tvorbu a konfiguraci GUI.
11. Proveďte rešerši v oblasti symbolických výpočtů v prostředí Scilab.
12. Proveďte objektivní a subjektivní srovnání práce v systémech MATLAB a Scilab.

Při objektivním hodnocení se zaměřte na rychlost spouštění obou systémů, rychlost srovnatelně rozsáhlých výpočtů, obtížnost debuggingu apod.
Při subjektivním hodnocení se zaměřte na intuitivnost celého systému (Popište, zda je systém jako takový User-friendly při běžné práci a při hledání a odstraňování chyb v skriptech a modelech.)

Práce bude vypracována v typografickém systému LaTeX.

Seznam doporučené odborné literatury:

[1] Stephen L. Campbell, Jean-Philippe Chancelier, Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos.* 2006. ISBN-10: 0-387-27802-8 ISBN-13: 978-0387278025.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 29.4.2016

Palenikova

Ráda bych na tomto místě poděkovala všem, kteří mi s prací pomohli, hlavně Ing. Janu Skapovi, Ph.D. za cenné rady a připomínky při vypracování mé bakalářské práce.

Abstrakt

Cílem této bakalářské práce je poskytnout uživatelům, kteří se zabývají problematikou zpracování signálů, volně dostupný výpočetní software. Z toho důvodu byla vytvořena podrobná uživatelská příručka práce v programu Scilab, který je bezplatnou alternativou licencovaného programu MATLAB. Tato práce také může pomoci uživatelům s přechodem mezi těmito systémy. Zároveň jsou zde probírány jednotlivé kapitoly, které se týkají zpracování signálů, kde jsou uvedeny ukázkové příklady s podrobným komentářem. U těchto příkladů jsou dále zdůrazněny rozdíly a odlišnosti obou systémů.

Klíčová slova: MATLAB, Scilab, SciNotes

Abstract

The goal of this bachelor thesis is to provide users, which are dealing with the issue of signal processing, open source computing software. For this reason user guide on how to work with the Scilab program was created, which is free alternative of licensed pogram MATLAB. This thesis can also help existing MATLAB users with transition between these two systems. At the same time there are individual chapters, that relate to signal processing, where there are listed examples with detailed description. These examples further highlight differences and diversity of these two systems.

Key Words: MATLAB, Scilab, SciNotes

Obsah

Seznam použitých zkratek a symbolů

2D	–	Two-dimensional
3D	–	Three-dimensional
ATOMS	–	AuTomatic mOdules Management for Scilab
GUI	–	Graphical User Interface

Seznam tabulek

Seznam obrázků

Úvod

Cílem této bakalářské práce je podrobně seznámit uživatele se systémem Scilab jako bezplatnou alternativou komerčního systému MATLAB. Také tato práce může pomoci uživateli s přechodem mezi těmito systémy. V současnosti se klade důraz na přesnost výpočtů, různých simulací a jejich vizualizaci, je proto potřeba mít k dispozici nějaký software, který umožňuje jednoduše a efektivně řešit tyto problémy. Dominantním softwarem na trhu je právě zmíněný MATLAB, jehož největší nevýhoda spočívá v tom, že je licencovaný a tudíž není k dispozici zdarma. Z tohoto důvodu byl vyvinut open-source software Scilab určený pro vědeckotechnické výpočty. Dále umožňuje provádět jednoduchou aritmetiku, ale i náročné výpočty, např. statické analýzy, zpracování obrazu nebo simulace fyzikálních a chemických jevů.

V první části práce se budeme věnovat instalaci programu na dva operační systémy - MS Windows verze 8 a Linux (distribuce Ubuntu). V dalším kroku se zaměříme na rozložení systému a možnosti konfigurace, dále na práci s konzolí a editorem SciNotes. Probereme si i práci s dokumentací (Help) a jednotlivé možnosti ladění programu. V další části se zaměříme na tvorbu a spouštění skriptů a funkcí. Popíšeme si možnosti tvorby 2D a 3D grafů včetně práce s osami v rámci okna „Figure“. Ukážeme si, jak se pracuje s externími soubory (textovými, binárními a zvukovými), jejich načítání, zápis a ukládání. Také se budeme věnovat možnosti efektivní výměny dat a převodem kódu mezi systémy MATLAB a Scilab. V neposlední řadě se podíváme na tvorbu a konfiguraci GUI včetně importů modulů pomocí správce ATOMS. Nakonec se zaměříme na rešerši v oblasti symbolických výpočtů. Zároveň se ve všech kapitolách budeme věnovat rozdílům práce mezi systémy MATLAB a Scilab.

1 Instalace systému Scilab

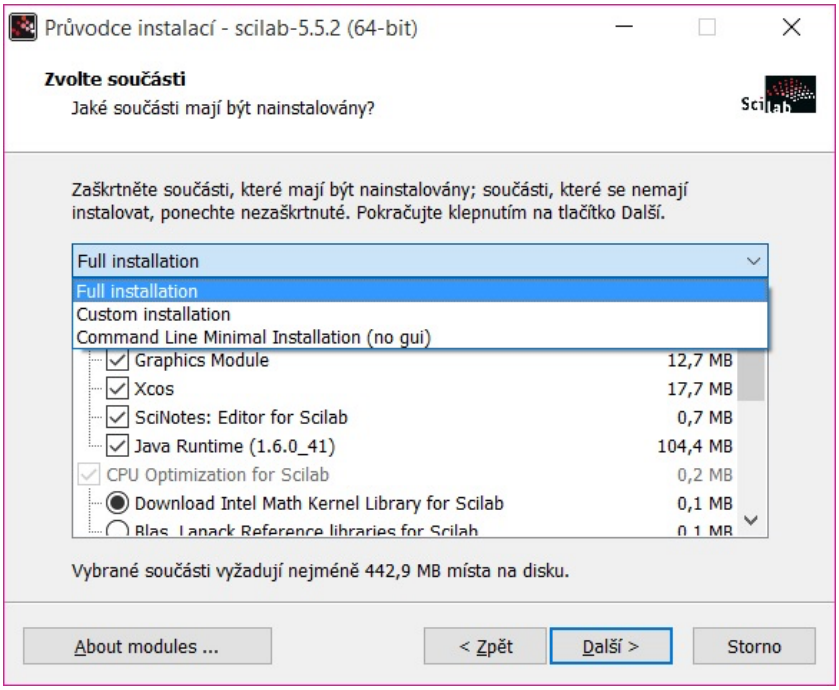
Scilab vznikl jako *open-source*, je dostupný pro všechny hlavní operační systémy (MS Windows, Linux, OSx). Lze jej stáhnout z domovské stránky Scilabu. Na této stránce nalezneme veškeré verze včetně variant pro různé platformy, jako jsou již zmíněny MS Windows (verze Vista, XP, 7, 8), Linux i OSx. V mé práci budeme rozebírat pouze instalace na MS Windows a Linux (distribuce Ubuntu). Nové verze Scilabu jsou publikovány zhruba jednou ročně. Nyní jsou k dispozici verze 5.5.2 a 6.0.0 Alpha 1, která je stále ve fázi testování (dostupná i pro MS Windows verze 10), proto se budeme věnovat stabilní verzi *Scilab 5.5.2*.

1.1 Instalace Scilabu ve Windows

Postup instalace závisí na tom, jakou máme verzi systému (32-bit nebo 64-bit). Verzi systému zjistíme tak, že na ikonu **Tento počítač** klikneme pravým tlačítkem myší a zvolíme **Vlastnosti**, poté se podíváme na **Typ systému**.

1.1.1 Instalace Scilabu

Spustíme stažený soubor s příponou `.exe`. Postupujeme podle pokynů instalačního průvodce, dokud se nedostaneme do bodu, kde máme možnost výběru typu instalace. Na výběr máme následující typy:



Obrázek 1.1: Výběr typu instalace

- **Plná instalace (Full installation)** - V tomto módu se nainstaluje JVM Module, který obsahuje Graphics Module, Xcos, SciNotes: Editor for Scilab a Java Runtime. Program stáhne matematickou knihovnu a numerický software FFTW. Dále nainstaluje MPI, TCL/TK a také vývojové nástroje. Pokud nevíme, k čemu přesně budeme program potřebovat, doporučuji plnou instalaci. Poté opět pokračujeme dle pokynů průvodce.
- **Vlastní instalace (Custom installation)** - U vlastní instalace je na nás, co si vybereme. Lze tedy nainstalovat vlastní kombinaci modulů.
- **Instalace bez uživatelského rozhraní (Command Line Minimal Instalation (no gui))** - Tento mód nám nainstaluje pouze příkazový řádek. Je zde možnost instalování rozšiřujících balíčků („toolboxů"). Jednotlivé balíčky jsou vzájemně provázány tzn. instalace jednoho z nich nemusí být možná pokud není nainstalován balíček další. Tyto souvislosti v helpu nejsou jednoznačně vysvětleny. Proto je velice problematické provést postupnou instalaci v minimální verzi Scilabu.

1.2 Instalace Scilabu v Linuxu (Ubuntu)

Nejdříve je nutné stáhnout aktuální verzi programu, která je k dispozici na domovské stránce Scilabu. Než stáhneme Scilab, je nutné vědět, jakou máme verzi systému. S ohledem na tuto verzi zvolíme odpovídající software Scilab 5.5.2 32-bit nebo Scilab 5.5.2 64-bit. Verzi systému zjistíme tak, že vpravo na horní liště klikneme na **Nastavení** → **O tomto počítači** → **Přehled**, nebo také pomocí příkazové řádky. Otevřeme si terminál pomocí klávesové zkratky **CTRL + ALT + T** nebo ikony **Hledání** → **Terminal** a napíšeme příkaz `name -u`.

1.2.1 Spuštění Scilabu

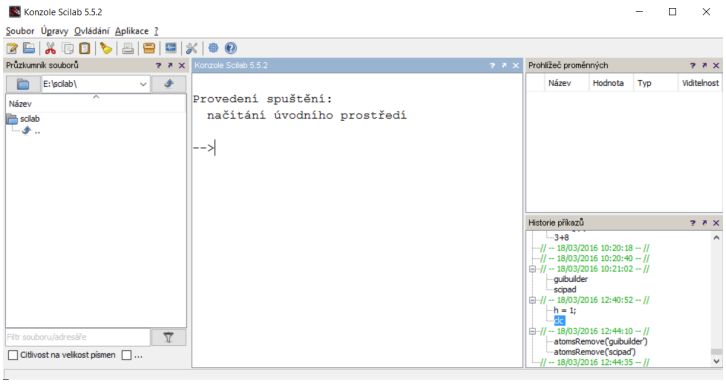
Stažený soubor extrahujeme do domovské složky uživatele. Poté otevřeme terminál, kde spustíme Scilab pomocí příkazu `~/scilab-5.5.2/bin/scilab`. Program se nám spustí a můžeme začít pracovat.

1.3 Instalace Scilabu v Linuxu (Ubuntu) pomocí balíčků

Tato instalace je dostupná pro verze *Ubuntu 14.04* a vyšší. Instalace pomocí správce balíčků je pro uživatele Ubuntu pohodlnější než manuální instalace. Do okna terminálu napíšeme příkaz `sudo apt-get install scilab`. Pokud se nám nepodaří nalézt balíček `scilab`, je potřeba provést příkaz `apt-get update`. Poté opět provedeme příkaz `sudo apt-get install scilab`.

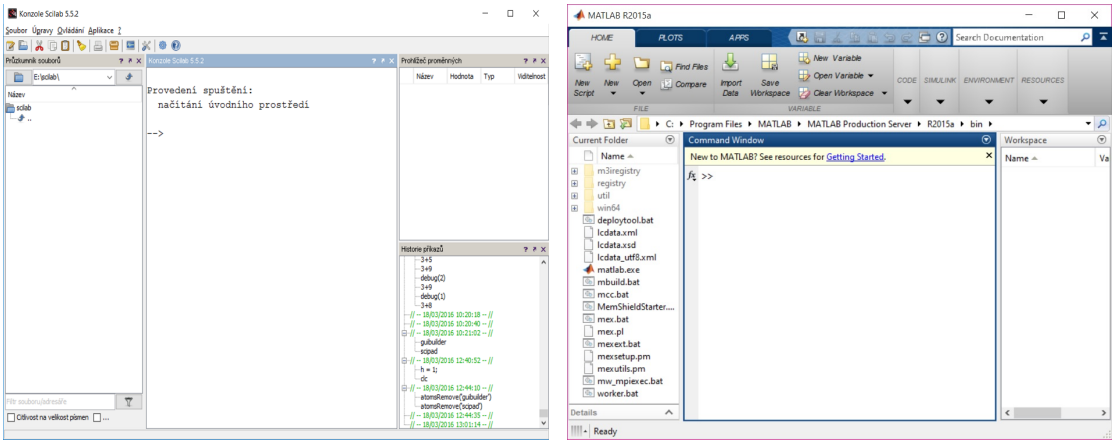
2 Prostředí systému Scilab

V operačním systému Windows program spustíme kliknutím na příslušnou ikonu. Po spuštění se otevře nové okno. Scilab pracuje s okenní strukturou, která je uživatelsky přívětivější než rozhraní příkazového řádku. Při prvním spuštění programu není součástí editor *SciNotes*.



Obrázek 2.1: Úvodní okno bez editoru SciNotes

V levé části programu se nachází **Průzkumník souborů**, kde se zobrazují podadresáře a soubory právě aktivního adresáře. V prostřední části je okno **Konzole** (příkazový řádek). V konzoli lze provádět jednoduché výpočty a výsledky se zobrazují přímo v ní pod příkladem. V pravé horní části se nachází **Prohlížeč proměnných**, kde jsou proměnné s hodnotami včetně typu, které jsme si nadefinovali v konzoli. Proměnné jsou různých typů (Integer, Double, String, ...). V pravé dolní části je okno **Historie příkazů**, kde je zobrazena historie příkazů. Příkazy jsou seřazeny podle toho, jak jsme je zadávali do konzole.

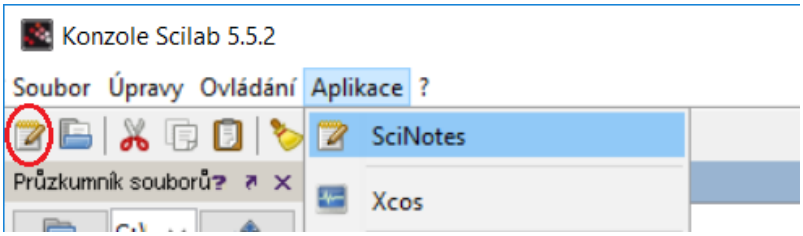


(a) Scilab (b) Matlab

Obrázek 2.2: Srovnání úvodních oken

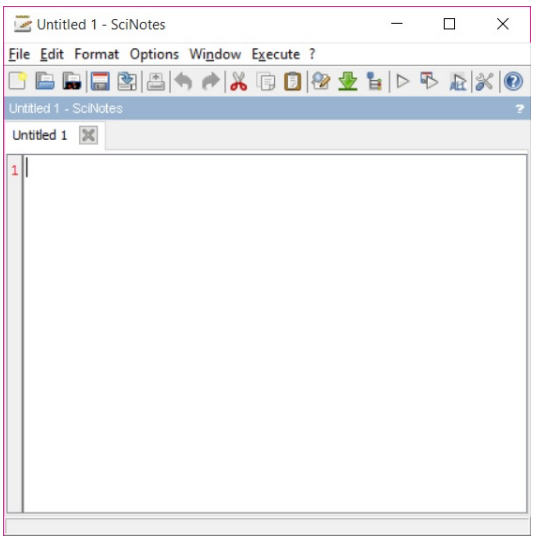
2.1 SciNotes

Editor *SciNotes* je nástroj k psaní kódu. Je propojen přímo s programem Scilab, tím umožňuje psát plnohodnotné programy, aniž bychom museli instalovat další podprogramy. Konzole nám stačí pouze pro jednoduché výpočty. Při každém ukončení programu se nám nejdříve ukončí editor a až poté samotný Scilab. Otevření editoru se může provést dvěma způsoby.



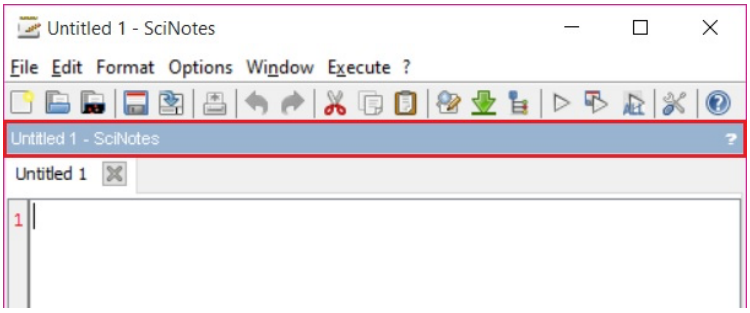
Obrázek 2.3: Otevření editoru SciNotes

Jeden ze způsobů je pomocí **ikony**, která se nachází na liště vlevo nahoře. Druhý způsob otevření editoru SciNotes je pomocí menu **Aplikace** → **SciNotes**. Po spuštění editoru se nám objeví samostatné okno.



Obrázek 2.4: Okno editoru SciNotes

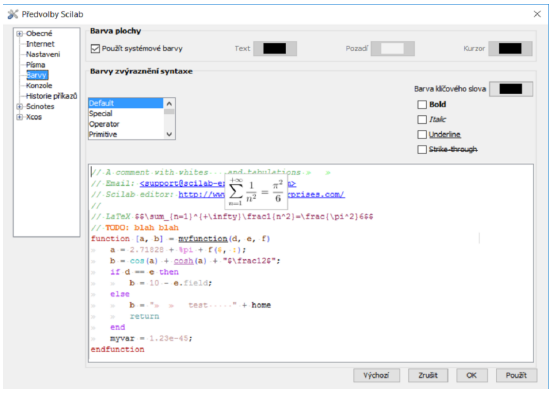
Máme dvě možnosti jak umístit editor, buď jako samostatné okno, jak je tomu automaticky po spuštění, nebo může být součástí hlavního okna. Umístění editoru do hlavního okna se provede následujícím způsobem: Otevřený editor chytneme za modrý pruh, kde se nachází název souboru, a přetáhneme ho na kterékoliv místo v hlavním okně. Nejprve se objeví obrys okna s náhledem umístění a teprve poté, co uvolníme okno, bude umístění skutečné. Po samotném umístění lze upravit výšku, případně šířku jakéhokoliv okna. To provádíme pomocí šipek, které jsou mezi jednotlivými okny. Po vypnutí programu se umístění oken uloží. Po opětovném zapnutí programu bude umístění shodné s vlastním nastavením.



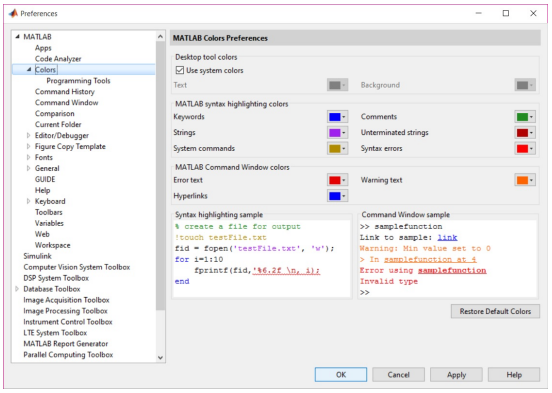
Obrázek 2.5: Připojení k hlavnímu oknu

2.2 Možnosti konfigurace

Scilab podobně jako MATLAB nabízí možnost změny barevného schématu pro zvýraznění syntaxe. Je tedy možné zcela jednoduše adaptovat barevné schéma MATLABu pro Scilab. Tyto změny můžeme provést v nastavení, které nalezneme na horní liště **Úpravy** → **Předvolby**.



(a) Scilab



(b) MATLAB

Obrázek 2.6: Srovnání oken pro nastavení barev

Pro dosažení požadovaného vzhledu postupujeme v souladu s popisem uvedeným v následujícím textu. MATLAB používá zelenou kurzívu pro komentáře a fialovou barvu pro řetězce. Pokud ve Scilabu chceme mít komentáře zelené, musíme si vyhledat pole **Comment** a vpravo si změníme barvu na zelenou. Následně zaškrtneme pole *italic*. Stejný postup provedeme pro řetězce. Vyhledáme si pole **String** a poté změníme barvu na fialovou.

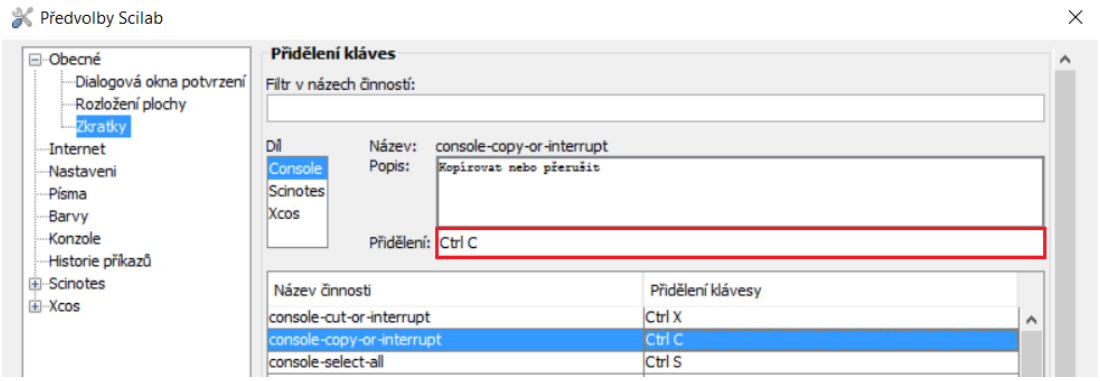
MATLAB nabízí mnohem více možností úpravy programu, aby byl uživatelsky přívětivější. Umožňuje změnu písma v Command Window, editoru i Command History. V Command History si můžeme nastavit písmo dle vlastního výběru, ale máme také možnost zvolit si typ písma stejný, jako je u desktop code nebo desktop text. Scilab povoluje změny písma pouze v konzoli a editoru.

Dále si ve Scilabu můžeme zvolit zkratky různých událostí. Základní nastavení nalezneme: **Úpravy** → **Předvolby** → **Obecné** → **Zkratky**. Zde můžeme nahlédnout a změnit zkratky v konzoli nebo v editoru SciNotes. Některé zkratky jsou vypsány v následující tabulce 2.1.

Tabulka 2.1: Zkratky v programu Scilab

Zkratky v konzoli	Význam
CTRL X	Vyjmutí nebo přerušení
CTRL C	Kopírování nebo přerušení
CTRL S	Vybrat vše
CTRL A	Začátek řádku
CTRL E	Konec řádku
CTRL B	Předchozí znak
CTRL D	Smazání dalšího znaku
CTRL H	Smazání předchozího znaku
CTRL K	Smazání všeho za kurzorem do konce řádku
CTRL U/ESCAPE	Smazání celé řádky
CTRL W	Smazání posledního slova

V nastavení si můžeme všimnout, že po kliknutí na jakoukoliv zkratku se nám vždy ukáže její popis a příslušná klávesová zkratka. Tu lze změnit tak, že si vybereme danou činnost a přidělíme ji novou zkratku. Zmíněný proces je znázorněn na následujícím obrázku 2.7.



Obrázek 2.7: Nastavení zkratek

Zvolili jsme zkratku s činností kopírovat, které odpovídá klávesová zkratka CTRL C. Tuto zkratku změníme na CTRL O a uložíme. Nyní musíme program vypnout a znovu zapnout, aby se načetlo nové nastavení. Pokud nastavení pouze uložíme a pracujeme dál, budou pro kopírování fungovat zkratky CTRL C a CTRL O.

V této kapitole jsem čerpala z literatury [2].

3 Práce v konzoli

Práce s konzolí je vhodná pro malé projekty či jednoduché příklady, jejichž rozsah je velmi malý. V případě větších projektů je vhodné použít SciNotes. Práce v editoru SciNotes se od práce v konzoli liší tím, že můžeme napsat nejprve celý kód a až poté jej spustit. Také syntaxe editoru je pestrá a barevná na rozdíl od konzole.

3.1 Základní operace v konzoli

Jednotlivé příkazy spouštíme klávesou *ENTER*. Příkazy se okamžitě provedou a výsledky se objeví na následujících řádcích. Pokud nechceme, aby se výsledek vypsal do konzole, ukončíme příkaz středníkem „`;`“. Očekává-li Scilab příkaz, pak jsou na začátku úvodní znaky `-->`.

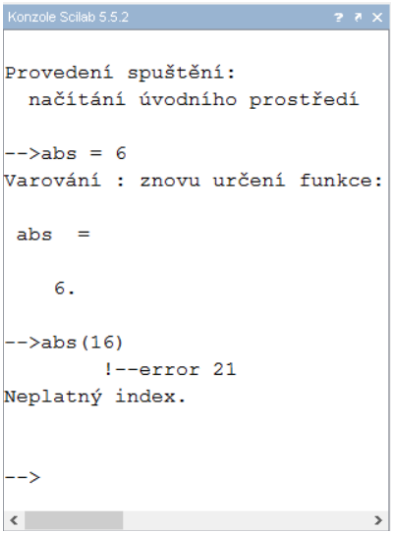
Proměnné mohou být až 24 znaků dlouhé. Při vytváření proměnné musíme dbát na jejich název. Ten musí začínat písmenem nebo znakem (_, %, $, #, ?). Scilab rozlišuje malá a velká písmena, tzn. `sloVO`, `Slovo` a `slovo` vidí jako tři odlišné proměnné.

Hodnoty do proměnných ukládáme pomocí operátoru `=`. Pokud zapisujeme hodnoty proměnných a neuvedeme název proměnné, do které se má hodnota uložit, Scilab ji automaticky uloží do proměnné `ans (answer)`.

Historie příkazů, které jsme zadali (od spuštění programu), se dají vyvolat šipkami nahoru a dolů. Proměnné se zobrazí v Prohlížeči proměnných. Tento prohlížeč lze vyčistit pomocí příkazu `clear();`, který nám vymaže všechny proměnné včetně jejich hodnot. Chceme-li vyčistit konzoli, použijeme příkaz `clc();`. Komentáře se ve Scilabu zapisují pomocí dvou lomítek // a končí na konci řádku. Vše, co je zapsáno za //, je bráno jako komentář, ale při běhu programu se komentáře ignorují. Nyní si ukážeme základní aritmetické operace. Mezi nejběžnější operace patří součet, rozdíl, součin, dělení a umocňování.

```
-->a = 6;
-->b = 4;
-->c = a + b
c  =

10.

-->d = a - b
d  =

2.

-->e = a * b
e  =

24.

-->f = a / d
f  =

3.

-->g = a ** b
g  =

1296.

-->g2 = a ^ b
g2  =

1296.
```

Při vytváření proměnných musíme dbát na to, abychom neukládali hodnoty do vestavěných funkcí. Na následujícím obrázku 3.1 je příklad vložení hodnoty do předdefinované funkce `abs`. Pokusíme se uložit hodnotu 6 do proměnné `abs`. Program nás upozornil, že je proměnná již funkcí, ale přesto nám hodnotu do proměnné zapíše. Tím předdefinovanou funkci přepíšeme. Když se poté pokoušíme získat absolutní hodnotu z čísla 16 použitím stejné funkce, program nám zahlásí `error 21 - Neplatný index`. Scilab proměnnou `abs` už nevidí jako funkci pro výpočet absolutní hodnoty. Přistupuje k ní jako k vektoru a snaží se získat prvek na pozici 16. V našem případě se v proměnné nachází pouze hodnota 16, nikoliv vektor. Z tohoto důvodu vrací chybu **neplatného indexu**.



Obrázek 3.1: Chybové hlášení

Scilab obsahuje speciální předdefinované konstanty, zapisují se pomocí %. Tyto konstanty jsou chráněné a nelze je vymazat. V následující tabulce 3.1 můžeme vidět výpis konstant a jejich ekvivalentní zápis v MATLABu.

Tabulka 3.1: Speciální konstanty v systému Scilab a MATLAB

Scilab	MATLAB	Popis
%i	i, j	imaginární jednotka
%e	e	Eulerovo číslo (e = 2.7182818)
%pi	pi	Ludolfovo číslo 3.1415927
%inf	inf	nekonečno
%nan	NaN	not-a-number
%t	logical(1)	true
%f	logical(0)	false
%eps	eps	maximální hondota

3.2 Základní operace s maticemi

Ve Scilabu je matice definována jako obdélníkové pole typu (m, n) obsahující data stejného typu (Integer, Double, String, ...) a uspořádána do m řádků a n sloupců. Proměnné mohou být číselnými hodnotami, vektory nebo maticemi. Vektory jsou považovány za matice typu $n \times 1$ nebo $1 \times n$, kde je n buď délka řádkového vektoru, nebo délka sloupcového vektoru. Matice se vkládají mezi hranaté závorky `[ ]`. Prvky v řádku matice se oddělují čárkou nebo mezerou. Prvky ve sloupci se oddělují středníkem „`;`“. Scilabem podporované operátory jsou uvedeny v tabulce 3.2.

Tabulka 3.2: Maticové operátory

Operátor	**Popis**
+	součet
-	rozdíl
*	násobení
/	pravé dělení
\	levé dělení
'	transpozice
^	exponent
.*	násobení jednotlivých prvků
.\	levé dělení jednotlivých prvků
./	pravé dělení jednotlivých prvků
.^	exponent jednotlivých prvků
.*.	násobení každý prvek s každým

Ukázka zápisu matice a řádkového i sloupcového vektoru:

```
-->A = [1 2 3 4; 4 5 6 7]
A =

 1.    2.    3.    4.
 4.    5.    6.    7.

-->B = [1 2; 4 5]
B =

 1.    2.
 4.    5.
```

```
-->a = [1 2 3 4]
a  =

 1.    2.    3.    4.


-->b = [1; 2]
b  =

 1.
 2.
```

Součet a rozdíl můžeme provádět jen pokud jsou vektory či matice stejného rozměru. Při sčítání a odčítání se vždy sečtou nebo odečtou prvky matice nebo vektoru se stejným indexem. Vytvoříme si matici `A`, matici `B`, vektor `v` a vektor `u`. Matice a vektory následně sečteme a odečteme.

```
-->A = [1 3 3 4; 4 9 6 7]
A =

1.    3.    3.    4.
4.    9.    6.    7.

-->B = [1 2 2 4; 5 5 6 7]
B  =

1.    2.    2.    4.
5.    5.    6.    7.

-->v = [1 6 8]
v =

1.    6.    8.

-->u = [2 5 7]
u =

2.    5.    7.
```

```
-->A + B
ans  =

2.    5.    5.    8.
9.    14.   12.   14.

-->A - B
ans  =

0.    1.    1.    0.
- 1.   4.    0.    0.

-->v + u
ans  =

3.    11.   15.

-->v - u
ans  =

- 1.   1.    1.
```

U matic se dá využít práce s dvojtečkou ':' v rámci jedné matice. Dvojtečka zastává více funkcí.

```
-->A = [1, 2, 3; 4, 5, 6];
-->B = A; B(1, :) = 20
B  =

20.   20.   20.
4.    5.    6.
```

```
-->C = A; C(1, 2:3) = 50
C  =

1.    50.   50.
4.    5.    6.
```

Nadefinovali jsme si matici `A` a `B`. Matice `B` byla vytvořena pomocí matice `A`. Vytvořenou matici jsme upravili tak, že v prvním řádku a v každém sloupci jsme zapsali číslo 20. První parametr závorky označuje řádek a druhý sloupec. Dvojtečka v tomto případě slouží jako označení všech sloupců, kde se uloží hodnota. Do matice `C` jsme ve sloupci 2 a 3 uložili hodnotu 50. Dvojtečka v tomto případě slouží jako rozsah od - do.

Využití symbolu `$` a jeho matlabovského ekvivalentu `end`. Tyto symboly mají význam výskytu posledního znaku.

```
-->D = A; D(1, $) = 100
D  =

1.    2.    100.
4.    5.    6.

-->E = A; E(1, 1:$) = 20
E  =

20.   20.   20.
4.    5.    6.
```

```
>> D = A; D(1, end) = 100
D =

1     2   100
4     5     6

>> E = A; E(1, 1:end) = 20
E =

20   20   20
4     5     6
```

V levé části je kód psaný ve Scilabu a v pravé části je kód psaný v MATLABu. U příkladu `D` jsme upravili matici pomocí symbolu `$`. V tomto případě tedy chceme v prvním řádku na poslední pozici zapsat číslo 100. V příkladu E jsme použili rozsah od - do. Uložili jsme tedy hodnotu 20 do prvního řádku na všechny pozice.

Další aritmetickou operací je násobení. Vytvoříme matici `A` a `B`, které budou mít stejný počet řádků a sloupců. Matice mezi sebou vynásobíme.

```
Konzole Scilab 5.5.2

-->A = [1 3 3 4; 4 9 6 7]
 A  =

    1.    3.    3.    4.
    4.    9.    6.    7.

-->B = [1 2 2 4; 5 5 6 7]
 B  =

    1.    2.    2.    4.
    5.    5.    6.    7.

-->A * B
      !--error 10
Nekonzistentní násobení.
```

Obrázek 3.2: Chyba při násobení matic

Při vynásobení jsme zjistili, že Scilab zahlásil **error 10**. Při násobení matic se postupuje tak, že první číslo řádku se vynásobí s prvním číslem sloupce. V našem případě má první matice v prvním řádku čtyři čísla a druhá matice ve druhém sloupci dvě čísla. Zbylé dvě číslice se nemají s čím násobit a program zahlásí chybu s názvem: **Nekonzistentní násobení**. Nastávají dvě situace: Matici `A` vynásobíme transponovanou maticí `B`, tedy výsledná matice bude ve tvaru 2×2 nebo můžeme matici `A` vynásobit bod po bodu maticí `B`. V tomto případě by se zachoval tvar matice.

```
-->A * B'
ans  =

 29.    66.
 62.    150.
```

```
-->A. * B
ans  =

 1.    6.    6.    16.
 20.   45.   36.   49.
```

Při násobení . $\times$. se násobí jednotlivé prvky matice mezi sebou, pokud máme matici 2×4, tak výsledná matice bude 4×16.

```
-->A. * .B
ans  =

 column 1 to 6

 1.    2.    2.    4.    3.    6.
 5.    5.    6.    7.    15.   15.
 4.    8.    8.    16.   9.    18.
 20.   20.   24.   28.   45.   45.

 column 7 to 12

 6.    12.   3.    6.    6.    12.
 18.   21.   15.   15.   18.   21.
 18.   36.   6.    12.   12.   24.
 54.   63.   30.   30.   36.   42.

 column 13 to 16

 4.    8.    8.    16.
 20.   20.   24.   28.
 7.    14.   14.   28.
 35.   35.   42.   49.
```

Při násobení řádkového vektoru a transponovaného vektoru je výsledkem **skalár**. Při násobení transponovaného vektoru a řádkového vektoru je výsledkem **matice**.

```
-->v = [1 6 8];
-->u = [2 5 7];

-->u * v'
ans  =

88.
```

```
-->u' * v
ans  =

2.    12.   16.
5.    30.   40.
7.    42.   56.
```

Další aritmetickou operací je dělení. U dělení platí pravidlo, že matice musí mít stejný počet sloupců.

```
-->A = [1 3 3 4; 4 9 6 7];
-->B = [1 2 2 4; 5 5 6 7];
-->C = [1 2 3; 4 5 6; 7 8 9];
-->A / B
ans  =

0.8142857   0.1571429
0.3428571   0.9714286

-->A / C
!--error 266
A a B musi mit stejny pocet sloupcu.
```

Dělení lze provést stejně jako u násobení, tedy po jednotlivých prvcích. Matice i vektory musí být stejné dimenze.

```
-->A. / B
ans  =

1.    1.5   1.5   1.
0.8   1.8   1.    1.

-->A. / C
!--error 9999
Inconsistent element-wise operation
```

```
-->v / u
ans  =

1.1282051

-->v. / u
ans  =

0.5   1.2   1.1428571
```

U příkladu, kde dělíme matici `A` maticí `C`, nám Scilab opět zahlásil chybu. Zde nám oznamuje, že matice nemají stejný počet řádků a sloupců.

Transponovaná matice je taková, která se otočí kolem hlavní diagonály a odebere se její přidaná část, tzn., že z matice 4×3 se stane 3×4.

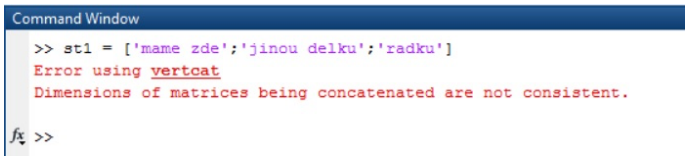
```
-->M = [1 2 3 4; 5 6 7 8; 9 10 11 12]
M  =

1.   2.    3.    4.
5.   6.    7.    8.
9.   10.   11.   12.
-->M'
ans  =

1.   5.   9.
2.   6.   10.
3.   7.   11.
4.   8.   12.
```

Funkce pro práci s maticemi:

- **zeros(m, n)** – nulová matice (všechny pozice jsou nulové)
- **ones(m, n)** – jedničková matice (všechny pozice jsou rovny jedné)
- **eye(m, n)** – jednotková matice (na hlavní diagonále jsou jedničky, jinak nuly)
- **rand(m, n)** – matice s náhodnými hodnotami mezi `m` a `n`
- **sum(x)** – provede součet všech prvků vektoru nebo matice
- **diag(x)** – vypíše všechny prvky ležící na hlavní diagonále
- **size(x)** - vypíše počet řádků a počet sloupců dané matice
- **det(x)** – výpočet determinantu, matice musí být čtvercová
- **inv(x)** – inverze čtvercové matice, ekvivalent této funkce je $\mathtt{x}^{-1}$
- **rank(x)** – určuje hodnost (max. počet lineárně nezávislých řádků) matice `x`
- **trace(x)** – provede součet čísel, které leží na hlavní diagonále
- **x = linspace(x, y, n)** – generuje aritmetickou posloupnost prvků `x` od `a` do `b`. Třetí parametr `n` je volitelný, udává počet prvků `x`. Parametr `n` je nastaven na 100, pokud není změněn na jiné číslo
- **x = logspace(x, y, n)** – generuje prvky `x` logaritmicky rozmístěné mezi body $10^{\mathtt{x}}$ a $10^{\mathtt{y}}$. Třetí parametr délky posloupnosti `n` udává počet prvků `x`. Parametr `n` je nastaven na 50, pokud není změněn na jiné číslo. Jeho ekvivalentem je $10^{\mathtt{linspace(x,\ y,\ n)}}$

Vytváření textových řetězců je další věc, kterou Scilab ovládá. Textové řetězce se vkládají do uvozovek (“) nebo též do apostrofů (‘). Jestliže chceme, aby řetězec obsahoval apostrof jako součást řetězce, musíme jej zdvojit, totéž platí i pro uvozovky.

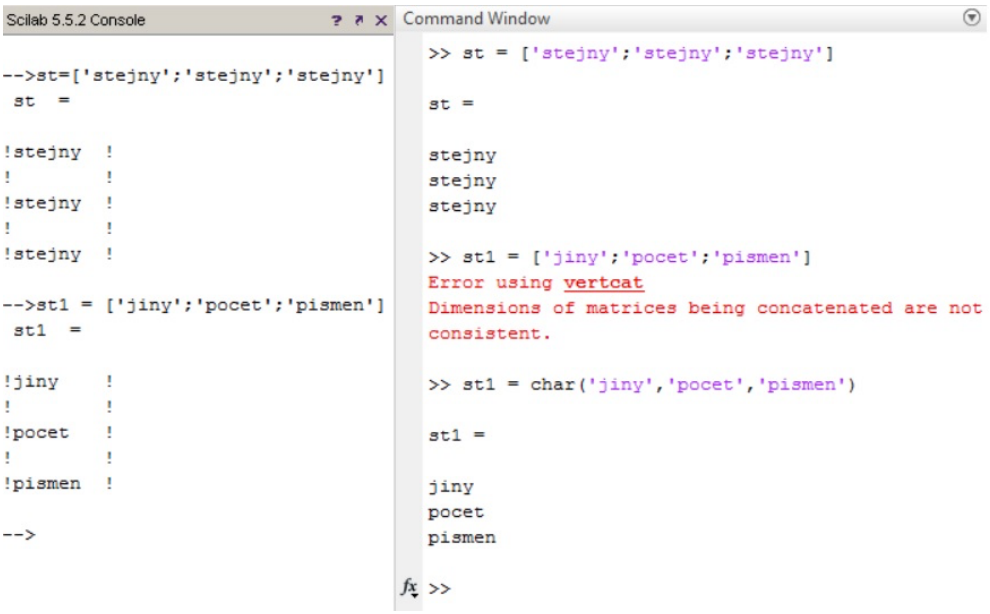
V MATLABu je možné skládat textové řetězce do matic - tedy jen za předpokladu, že všechny řádky matice jsou stejně dlouhé. V opačném případě nám MATLAB neumožní řetězce složit do matice.



Obrázek 3.3: Chybové hlášení v MATLABu

Tato chyba nám oznamuje, že matice není konzistentní, nebo-li pravidelná a souměrná. Musíme zadat řetězce o stejné délce. Řešení tohoto problému se skrývá ve dvou funkcích, které musíme použít, pokud chceme skládat řetězce jiné délky. Funkce `char()` a `str2mat()` doplní mezerami řádky kratší délky.

Ve Scilabu se také dají řetězce skládat do matic, ale není zde striktně dáno, že řádky matic musí být stejné délky. Scilab umožňuje skládat do matic jakkoliv dlouhé řetězce.



Obrázek 3.4: Srovnání řetězců ve Scilabu a v MATLABu

Ukázka výpisu velikosti řetězce ve Scilabu ve srovnání se zápisem v MATLABu.

```
-->STR = ['jedna'; 'dva'; 'tri'];

-->length(STR(2, :))
ans  =

3.
```

```
>> STR = char('jedna', 'dva', 'jedna');
length(STR(2, :))

ans =

5
```

Ve Scilabu je délka řetězce „dva“ rovna 3, avšak v MATLABu si můžeme všimnout, že je délka řetězce rovna 5. To je způsobeno tím, že v MALTABu je řádek doplněn mezerami na počet nejdelšího řádku v matici.

3.3 Základní operace s elementárními funkcemi

Elementární funkce je každá funkce, která vznikne jako výsledek konečného počtu operací sčítání, odčítání, násobení, dělení a skládání funkcí konstantních, mocninných, exponenciálních, logaritmických, goniometrických, tedy tzv. základních elementárních funkcí. Uveďme nyní stručný přehled těchto funkcí a jejich použití ve Scilabu.

3.3.1 Goniometrické funkce

Goniometrické funkce pracují s úhly a se stranami v pravoúhlém trojúhelníku. Mezi základní funkce patří sinus (`sin`), kosinus (`cos`), tangens (`tan`) a cotangengs (`cotg`). Sinus a kosinus jsou definovány pomocí odvěsen a přepony, tangens a cotangens pomocí odvěsen přilehlých a protilehlých. Ve Scilabu jsou předdefinované goniometrické funkce a inverzní goniomentrické funkce. Všechny se zapisují `y = f(x)`, kde `x` je vektor nebo matice.

Zápis (π) a jeho výpočty ve Scilabu:

```
-->sin(%pi/2)          -->cos(0)          -->tan(%pi/4)          -->cotg(%pi/4)
ans  =                 ans  =             ans  =                 ans  =

1.                     1.                 1.                     1.
```

Zápis (°) a jeho výpočty ve Scilabu:

```
-->sind(90)            -->cosd(0)         -->tand(45)            -->cotd(45)
ans  =                 ans  =             ans  =                 ans  =

1.                     1.                 1.                     1.
```

3.3.2 Logaritmické a exponenciální funkce

Přirozený logaritmus `ln(x)` se ve Scilabu zaznamenává jako `log(x)`, ale kromě přirozeného logaritmu Scilab umožňuje použití `log(x)` o základu 10 a 2. Zapisují se jako `log10(x)` a `log2(x)`.

```
-->log2(2)             -->log10(10)          -->log(1)
ans  =                 ans  =                ans  =

1.                     1.                    0.
```

Ve Scilabu nelze zapsat $\log_a(x)$, ale lze následně vypočítat: $\log_a(x) = \ln(x)/\ln(a)$. Přirozený logaritmus je inverzní funkcí exponenciální funkce `exp(x)`. Funkce se zapisuje `y = f(x)` = e^x. Tato funkce vrací hodnotu e^x pro reálná čísla `x`, `e` je základ přirozených logaritmů a Scilab má definovanou konstantu `e = 2,7182818`.

```
-->%e                                  -->exp(1)
%e  =                                  ans  =

2.7182818                              2.7182818
```

3.3.3 Mocniny a odmocniny

Program MATLAB nabízí několik funkcí pro práci s mocninami. Jednou z nich je `pow2(Y)`, kde `Y` je exponent. Tedy když zapíšeme `pow2(3)`, výsledkem je číslo 8 (2^3). Další funkce je `pow2(F,E)`, která vrací výsledek $F \times 2^E$. V neposlední řadě MATLAB nabízí funkci `power(n,m)`, kde `n` je číslo, které bude umocněno a parametr `m` je exponent. Ve Scilabu tato funkce neexistuje, ani zde není žádná ekvivalentní funkce, a máme tedy možnost pouze zdlouhavého zápisu. Uveďme si několik příkladů. V levé části jsou uvedeny příklady ze Scilabu a v pravé části z MATLABu.

```
-->x = 2^5

x  =

32.

-->x = 2 * 2^3

x  =

16.

-->x = 2^3

x  =

8.
```

```
>> pow2(5)

ans =

32

>> pow2(2,3)

ans =

16

>> power(2,3)

ans =

8
```

Pro odmocninu existuje v obou programech funkce `sqrt(x)`, která vypočítá druhou odmocninu z čísla `x`. Pokud se jedná o kladné číslo, výsledkem je kladné reálné číslo. Jestliže chceme získat odmocninu ze záporného čísla, výsledkem bude číslo komplexní.

3.3.4 Hyperbolické funkce

Hyperbolické funkce v matematice označují skupinu několika funkcí analogicky podobných k funkcím goniometrickým. Základní funkce jsou hyperbolický sinus (`sihn`) a kosinus (`cosh`), ze kterých je odvozen hyperbolický tangens (`tanh`) a kotangens (`coth`).

- $\sinh x = \frac{e^x - e^{-x}}{2} = \frac{e^{2x}-1}{2e^x}$
- $\cosh x = \frac{e^x + e^{-x}}{2} = \frac{e^{2x}+1}{2e^x}$
- $\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$
- $\coth x = \frac{\cosh x}{\sinh x} = \frac{e^x + e^{-x}}{e^x - e^{-x}} = \frac{e^{2x}+1}{e^{2x}-1}$

K těmto funkcím také existují funkce inverzní nebo-li hyperbolometrické funkce. Jedná se o argument hyperbolického sinu (`argsinh`), který se ve Scilabu zapisuje `asinh`. Dále argument hyperbolického cosinu (`argcosh`), argument hyperbolického tangens (`argtanh`) a argument hyperbolického cotangens (`argcoth`). Ve Scilabu se místo `arg` zapisuje pouze `a`, jak bylo uvedeno u argumentu hyperbolického sinu.

3.3.5 Komplexní čísla

Komplexní číslo se od běžných čísel liší především v tom, že obsahuje dvě části, a to část **reálnou** a **imaginární**. Je to dvojice uspořádaných čísel `[x,y]`, kde `x` představuje reálnou a `y` imaginární část. Pokud by `x` bylo nulové, jednalo by se o čistě imaginární komplexní číslo. Algebraický tvar je `x + y * i`, kde `i` se nazývá imaginární jednotka. Ve Scilabu je jako konstanta a zapisuje se `%i` .

Platí zde velmi důležitý vztah: $i^2 = -1$. Pokud bychom měli i^3, tak jej můžeme rozložit na $i^2 \times i$, kde i^2 víme, že je -1 a druhé i nijak nerozložíme, takže $i \times (-1)$ je $-i$. Podobně je tomu pro i^4, to můžeme rozložit na $i^2 \times i^2$, přičemž $i^2 = -1$, takže dostáváme $-1 \times (-1) = 1$.

Ve Scilabu existuje několik funkcí pro práci s komplexními čísly. Například funkce `real()`, která vrátí reálnou část čísla, `imag()`, která vrátí imaginární část čísla.

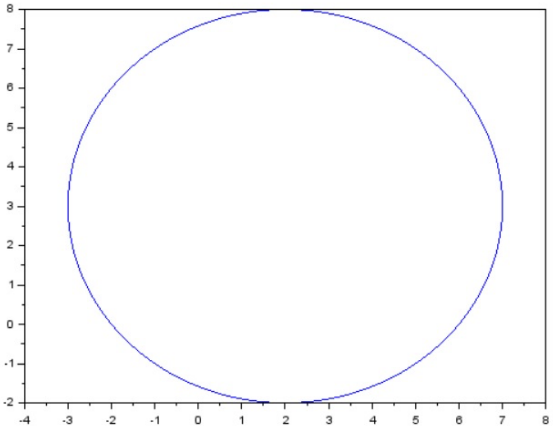
```
-->z = 5 + 4 *%i                -->rz = real(z)                 -->iz = imag(z)
z~=                             rz  =                           iz  =

 5. + 4.i                       5.                              4.
```

Absolutní hodnota komplexního čísla se zapisuje $|z| = \sqrt{x^2 + y^2}$. Ve Scilabu existuje funkce `abs(x)`. Dále určíme komplexně sdružené číslo pomocí funkce `conj(x)`

```
-->absz = abs(z)                -->conj(z)
absz  =                         ans  =

 6.4031242                      5. - 4.i
```

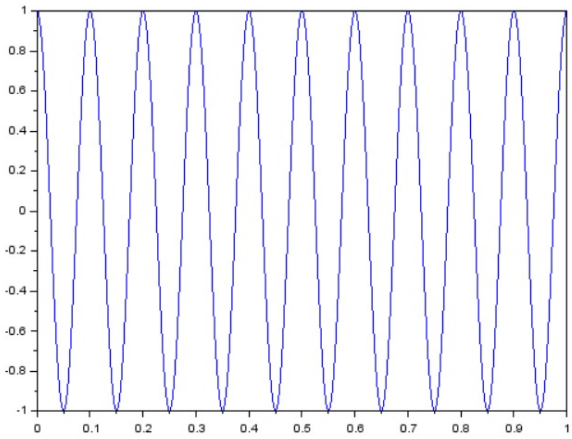
V MATLABu lze zkrátit vykreslení komplexních čísel pomocí `plot(z)`, tato možnost ve Scilabu není. Nyní si vykreslíme kružnici pomocí algebraického tvaru a komplexních čísel:

```
-->r = 5;
-->phi = 0 : %pi / 100 : 2 * %pi;
-->sy = 3;
-->sx = 2;
-->x = sx + r * cos(phi);
-->y = sy + r * sin(phi);
-->plot(x,y) // vykresleni pomoci algebraickeho tvaru

-->z = (sx + sy * %i) +r * exp(%i * phi);
-->plot(real(z),imag(z))//vykresleni pomoci komplexnich cisel
```



Obrázek 3.5: Vykreslení kružnice

```
-->vektor_t = 0 : 0.001 : 1;
-->x = cos(2 * %pi * 10 * vektor_t);
-->plot(vektor_t, x) // vykresleni cosinu pomoci vektoru
```



Obrázek 3.6: Graf funkce cos pomocí vektoru

Pro zjištění úhlu komplexního čísla existuje v MATLABu funkce `angle(x)`, která ve Scilabu není. Zjištění úhlu je možné pomocí funkce `atan(imag(A),real(A))`, kde se dosadí již zmíněné funkce pro zjištění reálného nebo imaginárního čísla.

```
-->atan(imag(z),real(z))
ans  =

0.6747409
```

3.3.6 Absolutní hodnota

Absolutní hodnota čísla je vždy nezáporné číslo. Pokud budeme chtít znát absolutní hodnotu z kladného čísla, dostaneme vždy číslo stejné. Absolutní hodnota záporného čísla, tedy čísla menšího než 0, bude absolutní hodnota rovna číslu opačnému. Funkcí pro vyjádření absolutní hodnoty je ve Scilabu `abs(x)`.

3.3.7 Zaokrouhlování a faktoriál

Ve Scilabu existuje několik funkcí pro zaokrouhlování:

- `ceil()` - zaokrouhluje směrem nahoru k nejbližšímu celému číslu
- `floor()` - zaokrouhluje směrem dolů k nejbližšímu celému číslu
- `round()` - zaokrouhluje podle matematických pravidel
- `fix()` - za desetinnou čárkou zaokrouhluje směrem dolů k nejbližšímu celému číslu

Pro výpočet faktoriálu existuje ve Scilabu funkce `gamma(x)`, která je definovaná jako `n! = gamma(n+1)`.

```
-->gamma(7+1)
ans  =

 5040.
```

3.4 Polynomy

Scilab umožňuje práci s polynomy čili s mnohočleny. Lze je vytvářet několika způsoby a práce s nimi je podobná práci s čísly. Polynomy jsou prezentovány jako vektory. Jedním ze způsobů vytváření polynomu existuje ve Scilabu příkaz `poly`. Při vytváření polynomů lze specifikovat koeficienty polynomu nebo jeho kořeny polynomu.

Syntaxe polynomu je `p = poly(a, vname, ["flag"])`, kde `a` je vektor, číslo nebo matice. Označení `vname` je proměnná a `flag` je řetězec, který má hodnotu `roots` nebo `coeff`. Pro získání kořenů polynomu použijeme hodnoty **`roots`** či **`coeff`** představující jeho koeficienty, které jsou uloženy v prvcích vektoru. První prvek vektoru představuje nejnižší koeficient rovnice a s každým následujícím prvkem se koeficient zvyšuje. Zápis polynomu bez a s předem definovaným vektorem:

```
-->p = poly([1 2 3],"x","coeff")
p  =

 2
1 + 2x + 3x
```

```
-->v = [1 2 5 3];
-->p = poly(v,"x","coeff")
p  =

2    3
1 + 2x + 5x + 3x
```

Pro určení kořenů polynomu ve Scilabu existuje funkce `roots(x)`, která vrací kořeny x_1 a x_2 ze zadaného mnohočlenu. Jestliže kořeny známe, ale potřebujeme určit polynom, tak vynecháme v syntaxi příkazu poslední atribut nebo použijeme parametr `roots`. Tedy `p = poly(a, vname, "roots")` nebo vynecháním atributu `p = poly(a, vname)`. Nyní si ukážeme zápis polynomu, zjištění kořenů polynomu a použití kořenů pro získání rovnice:

```
-->a = poly([2 3 1],"x","coeff")
a  =

2
2 + 3x + x

-->roots(a)
ans  =

- 2.
- 1.
```

```
-->y = poly([-1 -2], "x", "roots")
y  =

2
2 + 3x + x

-->y2 = poly([-1 -2], "x")
y2  =

2
2 + 3x + x
```

Druhý způsob, jak vytvořit polynom, je nadefinovat proměnnou `y` nebo jinou a následně jej zapsat v algebraickém tvaru pomocí dříve vytvořené proměnné.

```
-->y = poly(0,"y");
-->q = 1 + 2*y^2 + 3*y^3
q  =

2    3
1 + 2y + 3y
```

S polynomy můžeme manipulovat stejně jako s číslicemi, tzn. Scilab podporuje základní aritmetické operace mezi mnohočleny. Ty tedy můžeme sčítat, odečítat, násobit i dělit. Musíme si však dávat pozor, abychom měli stejnou proměnou, jinak nám Scilab zahlásí chybu. Pokud polynomy dělíme výsledkem je lomený výraz.

Uveďme si základní aritmetické operace s mnohočleny. Nejdříve si nadefinujeme dvě proměnné `p` a `p2`, poté můžeme provádět operace.

```
-->p = poly([2 3 5],"x","coeff");
-->p2 = poly([1 6 2 5],"x","coeff");

-->r = p + p2
r  =

2    3
3 + 9x + 7x + 5x

-->r2 = p - p2
r2  =

2    3
1 - 3x + 3x - 5x

-->r3 = p * p2
r3  =

2     3     4     5
2 + 15x + 27x + 46x + 25x + 25x

-->r4 = p / p2
r4  =

2
2 + 3x + 5x
---------------
2    3
1 + 6x + 2x + 5x
```

3.5 Nápověda

Nápovědu ve Scilabu lze vyvolat několika způsoby. Jedním ze způsobů je použití příkazového řádku. Do něj napíšeme `help` a stiskneme klávesu ENTER. Dále můžeme použít klávesu `F1` nebo ikonu `?`, která se nachází v menu. Na tuto ikonu najedeme myší a poté klikneme na `Nápověda Scilab F1`. Po použití jedné ze tří možností se otevře okno.

Obrázek 3.7: Nápověda

V levé části se nachází veškerá dokumentace s jednotlivými tématy, které jsou seřazeny systematicky. Pokud na kterékoliv téma klikneme, rozbalí se seznam příkazů relevantní k danému tématu. Pokud hledáme určitou funkci, můžeme ji vyhledat. Seznam témat se skrývá pod ikonou knihy. Hned vedle knihy se nachází lupa, která umožní hledat jednotlivé funkce. V pravé části se zobrazí podrobné informace k vybrané položce: Jednoduchá definice příkazu, vyvolávací sekvence, syntaxe příkazů, parametry, podrobnější popis příkazů, příklady a odkazy na podobné příkazy. Nápovědu můžeme využívat, ačkoliv zrovna nejsme připojeni k internetu, tedy funguje i offline.

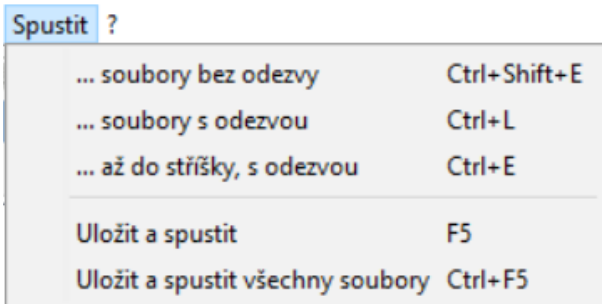
V této kapitole jsem čerpala z literatury [2], [3], [5], [13], [14].

4 Práce v editoru SciNotes

Důležitým prvkem ve Scilabu je možnost vytváření vlastních skriptů nebo funkcí. Rozdíl mezi skriptem a funkcí spočívá v tom, že funkce přijímá vstupní argumenty, ale skripty ne. Skripty i funkce lze velice jednoduše načíst do prostředí konzole pomocí příkazu `exec` nebo v nabídce **Soubor** → **Spustit**, kde s nimi můžeme dále pracovat. Jejich spuštění probíhá vždy v konzoli. Skript se ukládá s příponou `.sce` a funkce s příponou `.sci`.

Veškeré skripty nebo funkce se píší v editoru *SciNotes*. Scilab umožňuje také práci s programovacími nástroji, jako jsou podmínky a cykly. Program ve Scilabu je uspořádaná posloupnost instrukcí. Instrukce jsou příkazy, které říkají, co se má provést.

V editoru nespouštíme skripty (funkce) pomocí klávesy ENTER, ta slouží pro vložení nového řádku, musíme je tedy spustit pomocí tlačítka **Spustit**, které se nachází v toolbaru *SciNotes*. Zde máme několik možností spuštění.



Obrázek 4.1: Možnosti spouštění skriptů

- **soubory bez odezvy** - spustí soubor bez výpisu příkazů
- **soubory s odezvou** - spustí soubory i s výpisem jednotlivých příkazů
- **až do střížky, s odezvou** - spustí soubory jen do místa, kde se nachází kurzor včetně výpisu jednotlivých příkazů

4.1 Skripty

Skript, neboli procedura, obsahuje skupinu příkazů, které jsou samostatně uloženy. Jedná se o skupinu po sobě jdoucích příkazů, kde se používají i funkce. Výsledkem mohou být číselné výsledky i grafy. Zpravidla není volán funkcí nebo procedurou, ale i tato možnost není vyloučena. Do skriptů se mohou zapisovat i komentáře, což slouží pro větší přehlednost programu. Syntaxe příkazů a komentářů jsou barevně rozlišeny. Skript může pracovat s existujícími daty nebo může vytvářet data nová. Pro vytváření skriptů se využívá právě editor *SciNotes*. Doporučené požadavky pro zápis skriptu jsou následující:

- jméno souboru musí končit `.sce`
- jméno souboru by nemělo začínat číslicí, znakem nebo písmenem s diakritikou
- ve jméně souboru by neměla být použita mezera
- vždy začínáme skript komentářem, tzn. napíšeme // a popíšeme, co daný skript dělá a co je výsledkem
- píšeme komentáře za každý důležitý řádek, kde vysvětlíme význam daného příkazu

```
// skript, ktery vypocita objem a povrch koule
clc; // vymazeme konzoli

r = 5; // polomer koule
objem = 4/3 * %pi * r^3 // vypocet objemu koule
povrch = 4 * %pi * r^2 // vypocet povrchu koule
```

4.2 Funkce

Funkce obsahuje skupinu příkazů, které jsou samostatně uloženy včetně hlavičky a vstupních i výstupních parametrů. Funkce se volá jménem se skutečnými parametry, které se předávají do formálních podle pořadí. Syntaxe funkce musí mít následující tvar:

```
function <lhs_arguments> = <function_name> <rhs_arguments>
   <statements>
endfunction
```

- **<function_name>** - jméno funkce
- **<rhs_arguments>** - vstupní parametry, může se jednat o čárkou oddělenou sekvenci proměnných v závorkách, např. (x1,. . . ,xm). Poslední proměnná může být klíčové slovo *varargin* (proměnný seznam vstupních argumentů)

- **<lhs_arguments>** - výstupní parametry, může se jednat o čárkou oddělenou sekvenci proměnných v hranatých závorkách, např. [y1,. . . ,ym], kdy poslední proměnná může být klíčové slovo *varargout* (proměnný seznam výstupních argumentů)
- **<statements>** - tělo funkce

Níže je zobrazena ukázka funkce pro výpočet faktoriálu, na které si vysvětlíme základní pravidla tvorby funkcí a tuto funkci popíšeme. Definujeme funkci, která přijímá jeden parametr. Pomocí podmínky zajistíme vrácení správného výsledku pro číslo 0 a 1, abychom zaručili správný výpočet faktoriálu. Pro čísla vyšší než 1 provádíme cyklus `for`, který na začátku uloží do proměnné `i` číslo o 1 menší než vstupní parametr. Cyklus je dále prováděn, dokud je `i >= 1` a na konci každého průchodu se `i` zmenší o 1. V průběhu cyklu do vstupního parametru (`x`) postupně ukládá vstupní parametr vynásoben `i`. Po dokončení cyklu je vrácen výsledek prostřednictvím výstupní proměnné `x`.

```
// funkce pro vypocet faktorialu
clc;

function x = muj_faktorial(x)
if x <=1 then x = 1
else
for i = x-1:-1:1
x = x * i;
end
end
endfunction
```

Jednořádková funkce je skupina příkazů nebo funkcí, které se píší jako řetězec do jednoho řádku. Použití jednořádkových funkcí je výhodné pro jednoduché pomocné výpočty, které se vyskytují několikrát na různých místech. Například, pokud budeme potřebovat sečíst dvě hodnoty, zapíšeme danou funkci následovně: `deff('[vystup] = nazev_funkce(vstup1, vstup2)', 'vystup=vstup1+vstup2')`. V první části je název funkce s parametry a za čárkou (,) je v apostrofech uvedeno, co má daná funkce vykonávat. V tomto případě se sečte `vstup1` s `vstup2`.

```
-->deff('vystup=secti(vstup1,vstup2)','vystup=vstup1+vstup2');
-->secti(2,3)
ans  =

5.
```

4.3 Podmínky a cykly

Scilab nabízí podmínky `if-then-else` nebo `select-case` pro větvení programu. Poté cykly `for` a `while` pro vykonání nějaké činnosti opakovaně. Můžeme cyklit jeden příkaz nebo blok příkazů. Pro náhlé ukončení v bloku můžeme použít příkaz `break`, cyklus se ukončí a program pokračuje dále. Pro přerušení vykonávání cyklu můžeme použít příkaz `continue`, který po zavolání skočí na začátek cyklu.

4.3.1 Podmíněný příkaz (if-then-end)

Syntaxe příkazu: `if expr then statements else statements end`

Hlavním prvkem podmíněného příkazu je **podmínka**. Na základě této podmínky probíhá rozhodování, zda vykonat nebo nevykonat příkazy, které jsou součástí programu (`statements`). Vyhodnocování podmínek probíhá tak dlouho, dokud tento proces nevyhodnotí některou jako pravdivou. Další podmínky se poté nevyhodnocují. Pokud `if` logický výraz nevyhodnotí jako pravdivý, provede se tělo bloku za `else`, které se umisťuje na konec větvení. Můžeme větvit i pomocí `elseif`, který udává další podmínku k vyhodnocení. Jednoduchý program pro ověření, zda je číslo sudé nebo liché, by pak mohl vypadat následovně:

```
//ukazka vetveni if
clc;
n = 16;
if 0 == modulo(n,2) then
disp('Je sude')
else
disp('Neni sude, je liche')
end
```

Program nám vrátí výsledek: **Je sude**.

4.3.2 Přepínač (select-case)

Syntaxe příkazu: `select variable case value1 then instructions 1`
`... case valuen then instructions n [else instructions] end`

`Variable` je výraz, o kterém bude `select` rozhodovat. `Select` je ekvivalentem `switche`, který se používá v běžných programovacích jazycích. Hodnoty `value1, ..., valuen` jsou hodnoty u kterých se vyhodnocuje pravda nebo nepravda. `Instructions 1, ..., instructions n` jsou instrukce, které daný `case` provede. Na základě výrazu, který je za slovem `select`, přeskočí program na návěstí `case` se stejnou hodnotou, jakou má výraz `select`, a pokračuje vykonáváním příkazů za ním. Tento přepínač může obsahovat i `else`, pokud výraz neodpovídá žádnému návěstí. Níže si ukážeme jednoduchý příklad použití příkazu `select`.

```
//ukazka vetveni case
clc;
n = 2;
select n
case 1 disp('Cislo 1')
case 6 disp('Cislo 6')
case 5 disp('Cislo 5')
else disp('Je to jine cislo')
end
```

Program nám vrátí výsledek: **Je to jine cislo**.

4.3.3 Cyklus for

Syntaxe příkazu: `for var=expr, instruction, ..., instruction, end`

`Var` je proměnná, `expr` je výraz a `instruction` jsou jednotlivé instrukce, které se budou vykonávat. Po zpracování instrukcí je cyklus ukončen příkazem `end`. Cyklus `for` nám umožňuje opakovat určitou činnost po přesně definovaný interval (n-krát). Je vhodné jej používat v případech, kdy generujeme data nebo procházíme matice či vektory. Vždy však potřebujeme znát **počet opakování**, pokud ho neznáme, musíme použít řídící strukturu `while`. Na následující ukázce procházíme cyklus od 1 do 2.

```
//ukazka cyklu for
clc;
n = 1;
for i=1:2 // pokud je i rovno 1 az 2
n = n + 1 // navysime n o 1
end
disp(n)
```

Program nám vrátí výsledek: **3**.

4.3.4 While

Syntaxe příkazu: `while expr, instructions, ...[,else instructions], end`

`Expr` je výraz (expression), `instructions` jsou instrukce, jenž se mají provést a `end` značí konec cyklu while. Cyklus while je podobný příkazu `for`, ale na rozdíl od `for` není přesně definován konec cyklu. Cyklus se opakuje, dokud není podmínka splněna. Nevýhodou cyklu `while` je, že pokud špatně nastavíme podmínku, může to vést k zacyklení celého programu. Následující příkaz provádí přičítání `1` k `n`, dokud je `n < 4`:

```
//ukazka cyklu while
clc;
n = 0;
while n<4 //dokud je n mensi nez 4
n = n + 1; //navysime n o 1
end
disp(n)
```

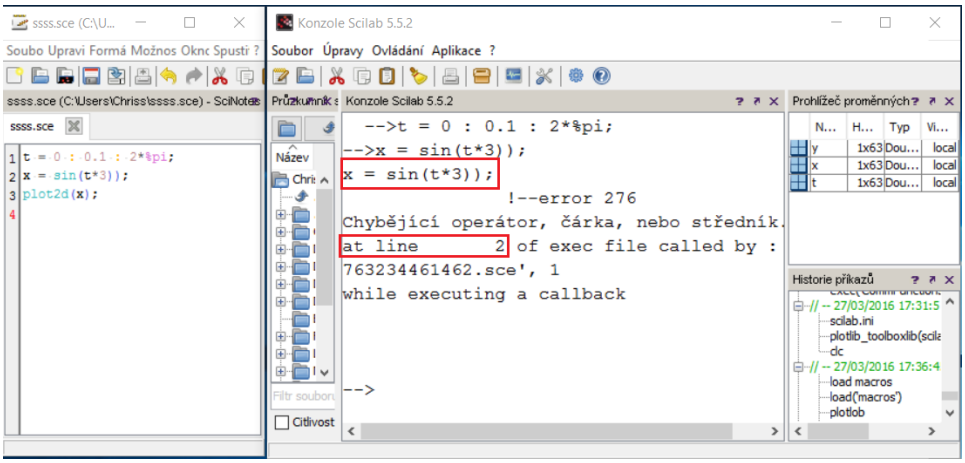
Program nám vrátí výsledek: **4**.

4.4 Ladění zdrojového kódu (debugging)

Během psaní zdrojového kódu se může stát, že uděláme chybu. U menších programů, které nejsou obsáhlé, není velkým problémem tuto chybu nalézt. U rozsáhlejších projektů je mnohem složitější se chyb dohledat. V předchozích verzích Scilabu se k ladění zdrojového kódu využíval editor Scipad. Ten však v nejnovějších verzích (5.1.1 a výše) není k dispozici. Scipad byl odstraněn z důvodů radikálních změn ve zdrojovém kódu Scilabu a odstranění programových API, na kterých tento editor závisí. V novějších verzích je k dispozici prostřednictvím správce balíčků ATOMS, nicméně jeho funkcionalita je značně omezena, kdy například grafický debugger nefunguje. Proto si v téhle kapitole povíme o možnostech debuggingu bez ladícího programu. Jednou z možností je vyhledat chyby pomocí chybových zpráv, které se nám objeví v konzoli. Dále pomocí funkcí, které zapisujeme do samotného kódu. Tyto funkce jsou součástí samotného programu.

4.4.1 Ladění zdrojového kódu pomocí chybových zpráv

První možnost, jak odladit program, je pomocí chybových zpráv. Pokud někde v projektu uděláme chybu, program zahlásí, kde jsme chybu udělali a čeho se týká. Také nám vypíše, na kterém řádku jsme chybu udělali.



Obrázek 4.2: Ukázka ladění zdrojového kódu pomocí výpisu chyb

Na obrázku můžeme vidět, že nám program zahlásil chybu, která nám oznamuje, co jsme udělali špatně a na kterém řádku se nachází. Podle hlášení jsme udělali chybu na druhém řádku v příkaze `x = sin(t*3))`. Můžeme si všimnout, že u příkazu je závorka navíc, proto tuto závorku vymažeme. Poté by měla chyba znovu po spuštění zmizet. Můžeme také nahlédnout na internetové stránky nápovědy, kde jsou vypsány všechny chybové zprávy a jejich význam.

4.4.2 Ladění zdrojového kódu pomocí funkcí

Další možností je využít funkce, které Scilab nabízí. Máme několik funkcí určených výhradně pro debugging, nastavení breakpointu, zobrazení breakpointu, smazání breakpointu, pozastavení běhu programu a další. V následujících podkapitolách si jednotlivé funkce uvedeme.

Funkce debug

Syntaxe příkazu: `debug(level_int); level_int = debug()`

Pro hodnoty 0, 1, 2, 3, 4 proměnné `level_int`, funkce definuje různé úrovně ladění. Používá se pro parser, nikoliv pro skripty, a je určena pouze Scilab expertům.

Funkce setbpt

Syntaxe příkazu: `setbpt(macroname [,linenumb])`

Tato funkce interaktivně vkládá breakpointy na dané číslo řádku (argument `linenumber`, jehož výchozí hodnota se rovná 1). Argument `linenumber` může nabývat hodnot řádkového nebo sloupcového vektoru, čísel řádků nebo jednoduchého čísla. Čísla řádků jsou rovna fyzickým číslům řádků uvnitř funkce `macroname`.

Jakmile Scilab při zpracování programu narazí na **breakpoint**, vykoná příkazy na tomto řádku a zastaví zpracovávání. Pokud funkce není kompilována, vypíše se tento řádek do konzole. V tomto bodě se Scilab nachází v „pozastaveném“ módu, kde uživatel může zkontrolovat hodnoty aktuálních proměnných či pokračovat ve vykonávání pomocí příkazů `resume` nebo `abort`.

Změna implementace funkce automaticky nesmaže dříve definované breakpointy, je tedy na uživateli je smazat pomocí funkce `delbtp`. Maximální počet funkcí, u kterých můžeme najednou nastavit breakpointy, musí být menší než 100 a celkový maximální počet samotných breakpointů je nastaven na hodnotu 1000.

Funkce dispbpt

Syntaxe příkazu: `dispbpt()`

Breakpointy nastavené pomocí dříve popsané funkce `setbpt` lze zobrazit funkcí `dispbpt`. Ta nepřijímá žádný argument a po jejím zavolání zobrazí všechny aktuálně nastavené breakpointy. Čísla řádků zobrazených touto funkcí jsou fyzická čísla řádků v rámci laděných funkcí.

Funkce delbpt

Syntaxe příkazu: `delbpt([macroname [,linenumb]])`

Smaže breakpoint definovaný na daném řádků `linenumber` ve funkci `macroname`. Podobně, jako tomu bylo ve funkci pro nastavení breakpointů, i v tomto případě může být argument `linenumber` řádkový či sloupcový vektor čísel řádků nebo jednoduché číslo.

V případě vynechání parametru `linenumber` jsou smazány všechny breakpointy uvnitř funkce zadané v parametru `macroname`. Pokud zavoláme tuto funkci bez parametrů, smažou se všechny breakpointy uvnitř všech laděných funkcí.

```
// funkce pro vypocet faktorialu
clc;

function x = muj_faktorial(x)
if x <=1 then x = 1
else
for i = x-1:-1:1
x = x * i;
end
end
endfunction

setbpt('muj_faktorial',[2, 5]) //nastavime breakpointy na radek 2 a 5
dispbpt() // zobrazime nastavene breakpointy
muj_faktorial(4) //spusteni funkce
delbpt('muj_faktorial',2) // smazeme breakpoint na radku 2
delbpt() // smazene vsechny aktualne nastavene breakpointy
```

Výše uvedený příklad popisuje možnosti použití funkcí pro práci s breakpointy. V tomto příkladu jsme použili funkci pro výpočet faktoriálu, kterou jsme si dříve vytvořili v kapitole 4.2. Nastavili jsme breakpointy pomocí funkce `setbpt` na řádek 2 a 5 a ty jsme si následně nechali zobrazit pomocí funkce `dispbpt`.

```
Body preruseni funkce : muj_faktorial

2
5
Zastaveni po radku   5 ve funkci muj_faktorial.

-1->
```

V tomto bodě se zastavilo vykonávání programu na řádku 5, kde můžeme vypsat hodnoty aktuálních proměnných, pokračovat ve vykonávání pomocí příkazu `resume`, nebo ukončit ladění pomocí příkazu `abort`. Po dokončení ladění programu smažeme breakpointy, které můžeme smazat jednotlivě, nebo všechny najednou.

Funkce where

Syntaxe příkazu: `[linenum, callername] = where()`

Tato funkce vrací vektory `linenum` a `callername`. Ty říkají, že dané instrukce byly zavolány řádkem `linenum(1)` funkce `callername(1)`), `callername(1)` byla zavolána řádkem `linenum(2)`) funkce `callername(2)`), a tak dále.

`callername(i)` je obecné jméno funkce, ale může být roven i `exec` nebo `execstr`, pokud se instrukce nachází uvnitř exec souboru nebo se jedná o execstr instrukci.

Funkce whereami

Syntaxe příkazu: `whereami()`

Zobrazí strom volaných instrukcí, které obsahují funkci `whereami()`. Může být použita v rámci pozastaveného módu.

V této kapitole jsem čerpala z literatury [1], [2], [4], [6], [12].

5 2D grafy

Scilab poskytuje možnosti pro velice rozsáhlou práci s grafikou, tvorbou 2D a 3D grafů. Nabízí také práci s více druhy speciálních grafů. Grafické prostředí má hierarchickou stromovou strukturu. Vrcholem je *Figure* (obrázek, graf, ...) a každé grafické okno má alespoň jedno *Dítě* (child) typu *Osa* (axe). Každý objekt typu *Osa* zase obsahuje sadu grafických objektů, jako jsou křivky a úsečky. Ve Scilabu můžeme vytvářet nejen 2D, ale také 3D grafy. 2D grafy jsou v programu tvořeny dvěma funkcemi: `plot` a `plot2d`. Tyto dvě funkce fungují na stejném principu, ale syntaxe a zadávání parametrů vypadá jinak. Funkce `plot` je převzatá z programu MATLAB včetně syntaxe kvůli zlepšení kompatibility s programem Scilab. Funkce `plot2d` je původem ze Scilabu.

5.1 Funkce plot

Nyní se zaměříme na funkci `plot`. Syntaxe příkazu: `plot(x,y,<LineSpec>,<GlobalProperty>)`, kde `x`, `y` jsou čísla nebo vektory. Ostatní parametry nejsou povinné. Můžeme vynechat například `x` nebo `y` podle toho, co se snažíme vykreslit.

`LineSpec` se používá pro nastavení barev nebo různých stylů grafu. Zapisuje se jako řetězec a vpisuje se přímo do funkce `plot`, kde změní definovanou křivku. Upravuje barvy, styl čar nebo značky. Na následující tabulce si ukážeme další možnosti nastavení.

Tabulka 5.1: Nastavení barev pomocí LineSpec

Znak	**Význam**	**Znak**	**Význam**
r	červená	**-.**	čárkovano-tečkovaná čára
g	zelená	+	značka plus
b	modrá	**o**	značka kolečko
c	tyrkysová	*	značka hvězdička
m	fialová	**.**	značka bod
y	žlutá	**x**	značka křížek
k	černá	**'square' or 's'**	značka čtverec
w	bílá	**'diamond' or 'd'**	značka kosočtverec
-	spojitá čára	^	značka trojúhelníku nahoru
–	čárkovaná čára	**v**	značka trojúhelníku dolu
'pentagram' or 'p'	pentagram	<	značka trojúhelník doleva
:	tečkovaná čára	>	značka trojúhelníku doprava

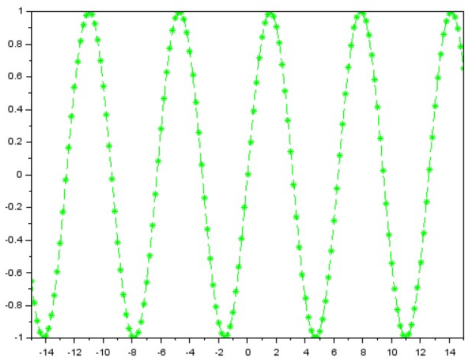
`Global Property` (globální parametry) nastavuje vlastnosti pro všechny křivky, které jsou definovány najednou. Tyto parametry se zadávají ve dvojici `PropertyName` a `PropertyValue`. První parametr dvojice musí být řetězec, jež definuje vlastnost, kterou chceme nastavit. Druhý parametr může být reálné číslo, řetězec, skalár nebo matice. Záleží na tom, kterou vlastnost jsme zadali v `PropertyName`. Hodnoty parametrů jsou zobrazeny v tabulce.

Tabulka 5.2: Nastavení parametrů pomocí Global Property

Znak	Význam
marker	značka
markforeground	barva značky
markbackground	barva výplně značky
markersize	velikost značky
color, colo	barva
linest, linestyle	styl čáry
thickness	tloušťka čáry

Vytvoříme si graf funkce `y = sin(x)`, zvolíme si barvu čáry například zelenou, čárkovanou čáru a nakonec značku hvězdičky

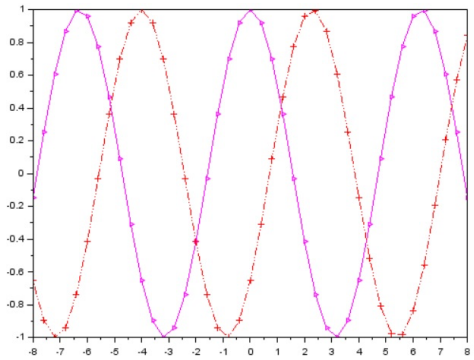
```
x = -15:0.2:15;
plot(x,sin(x),'g--*')
```



Obrázek 5.1: Graf funkce sin(x)

Ve Scilabu je možné vkládat více funkcí do jednoho grafu. Vložíme dvě funkce `y = cos(x)` a `y = cos(x+4)` do jednoho grafu. Každému nastavíme jiné vlastnosti (barvu, značku, barvu značky).

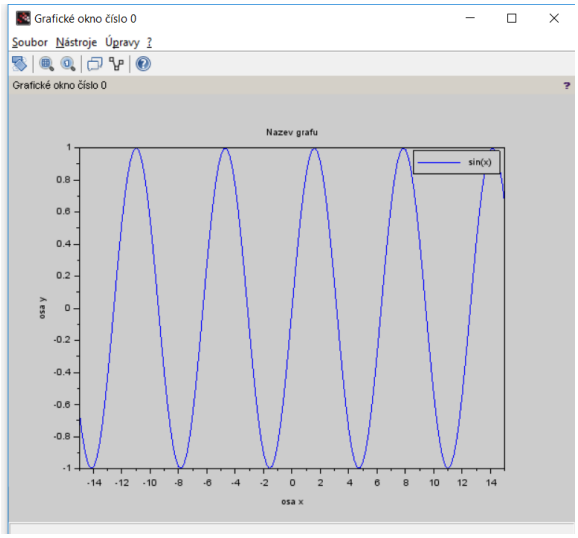
```
x =-8:0.4:8;
plot(x,cos(x),'m->', x, cos(x+4), 'r:+')
```



Obrázek 5.2: Graf funkce cos(x) a cos(x+4)

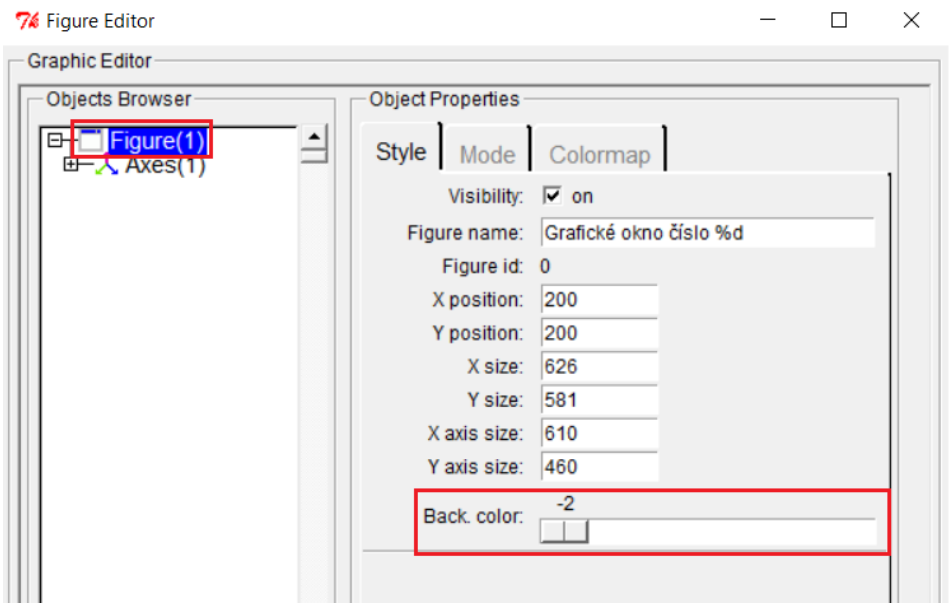
Při vytváření grafů můžeme využít okno *Figure*, kde doplníme popisky os, legendu a název grafu.

```
x = -15:0.2:15;
figure();
plot(x, sin(x));
xlabel("osa x");
ylabel("osa y");
title("Nazev grafu")
legend("sin(x)")
```



Obrázek 5.3: Ukázka grafu v okně Figure

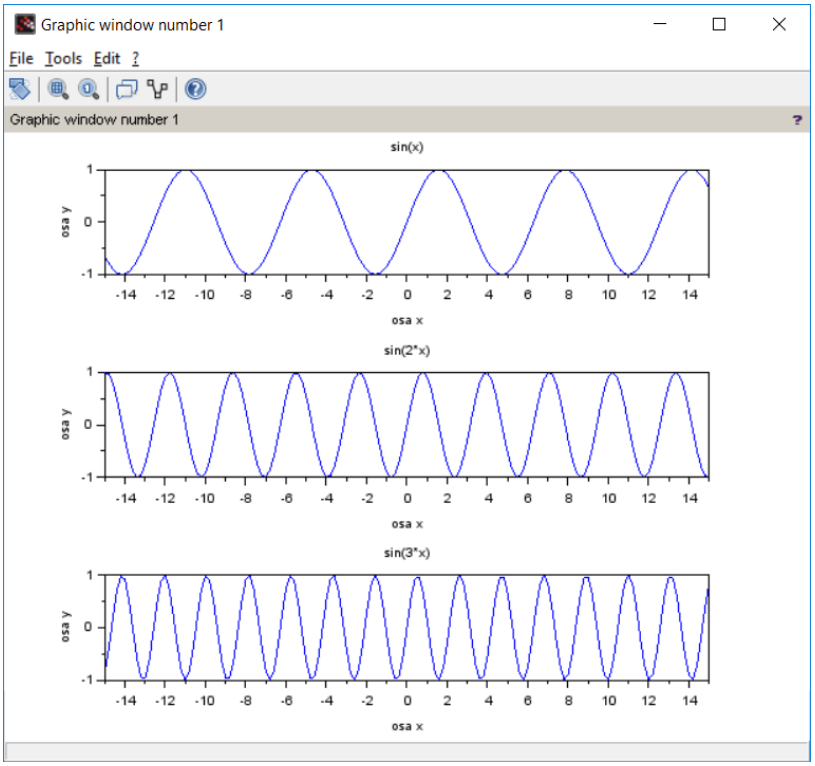
Barvy pozadí při vykreslení pomocí okna Figure a při vykreslení bez okna se liší. Pokud chceme, aby barvy pozadí byly stejné, klikneme v okně Figure na **Úprava** → **Vlastnosti obrazce**, kde si označíme celé okno Figure a změníme barvu pozadí na bílou. Nastavení je pouze pro dané okno, při spuštění nového okna je potřeba znovu přenastavit barvu.



Obrázek 5.4: Změna pozadí okna Figure

Po nastavení barevného schématu okna Figure máme možnost vložit více grafů do jednoho okna. Jejich rozložení v okně záleží na nastavení v `subplot()`. Vytvoříme si několik grafů, které vložíme pod sebe. Použijeme `subplot()`, kde první parametr udává číslo o kolik grafů se jedná, v našem případě je to číslo 3. Dále následuje číslo, které udává, zda máme grafy pod sebou, vedle sebe nebo za sebou. Vytvoříme grafy pod sebe, proto zvolíme číslo 1. Poslední parametr udává číslo pořadí.

```
x = -15:0.2:15;
figure(1);
subplot(3,1,1);
plot(x, sin(x));
xlabel("osa x");//popis osy x
ylabel("osa y");//popis osy y
title("sin(x)");//nazev podgrafu

subplot(3,1,2);
plot(x, sin(2*x));
xlabel("osa x");
ylabel("osa y");
title("sin(2*x)");

subplot(3,1,3);
plot(x, sin(3*x));
xlabel("osa x");
ylabel("osa y");
title("sin(3*x)");
```



Obrázek 5.5: Ukázka několika grafů v okně Figure

5.2 Funkce plot2d

Nyní se přesuneme k `plot2d`, která je původem scilabská funkce. Syntaxe příkazu: `plot2d([logflag,][x,],y,<opt_args>)`, kde `opt_args` umožňuje nastavit parametry grafu. Pokud bychom chtěli vykreslení grafu se vším všudy, použijeme syntaxi `plot2d([logflag,][x,], y[,style[,strf[,leg[,rect[,nax]]]]])`. Nyní si ujasníme jednotlivé parametry. Proměnné `x` a `y` jsou vektory nebo matice. Nastavitelné parametry jsou: `logflag, style, strf, leg, rect, nax, frameflag` a `axesflax`.

- **`logflag`** definuje typ měřítka pro každou osu (lineární nebo logaritmické). Hodnoty se zapisují pomocí řetězců. Ty mohou nabývat hodnot `nn`, `nl`, `ln` a `ll`, kde `n` je normální měřítko (lineární) a `l` je logaritmické.

- **`style`** nastavuje styl vykreslení čáry grafu. Hodnota je vektor, který obsahuje čísla kladná i záporná. Pokud obsahuje kladné číslo, má čára barvu podle palety čísla. Pokud je číslo záporné nebo nulové, místo čáry se vykreslí značka, která odpovídá zadanému číslu.

- **`strf`** řídí zobrazení titulků, řetězec o délce 3 `strf="xyz"`, kde ve výchozím nastavení je `strf="081"`, `x` může nabývat hodnot 0 nebo 1, `y` může nabývat hodnot od 0 do 8 a `z` od 0 do 5.

- **leg** nastavuje legendu, která se zobrazí pod grafem. Hodnotou je řetězec znaků, který má podobu "legenda1@legenda2", kde legenda1 přísluší první křivce grafu a legenda2 druhé křivce. Legendy se oddělují pomocí @. Vypisují se pod osu x. Tento parametr není jediný, který slouží k zobrazení legendy. Existuje funkce legend, která má více možností a je flexibilnější.

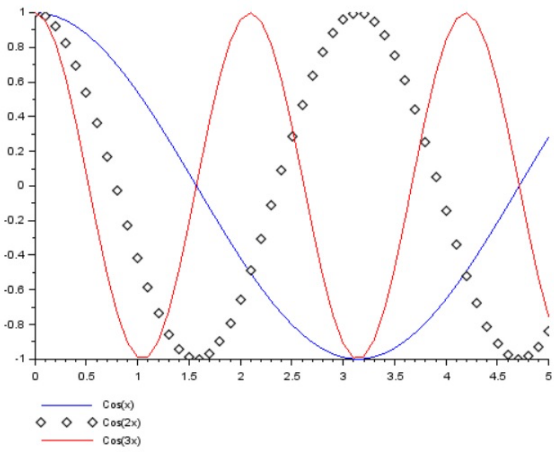
- **rect** definuje obdélník s limitními hranicemi pro graf. Hodnota parametru je vektor obsahující 4 položky [xmin, ymin, xmax, ymax]. Tento argument může být používán společně s frameflag, který specifikuje, jak jsou získány meze z rect.

- **nax** definuje dílkování stupnice každé osy ve formě vektoru čísel [nx, Nx, ny, Ny]. Parametr nx (ny) je počet dílků stupnice na ose x (y). Nx (Ny) je počet dílku mezi dvěma dílky stupnice.

- **axesflag** určuje možnosti vykreslení jednotlivé osy, jedná se o celá čísla v rozsahu 0 až 5. Výchozí nastavení axesflag je 9.

Nyní si ukážeme příklad s použitím některých parametrů. Vytvoříme si graf funkce y = cos(x), y = cos(2×x) a y = cos(3×x). Po všechny křivky zobrazíme legendu (leg) a styl čáry (style).

```
x=[0:0.1:5]';
plot2d(x,[cos(x) cos(2*x) cos(3*x)], leg="Cos(x)@Cos(2x)@Cos(3x)", style = [2,-5, 5] )
```



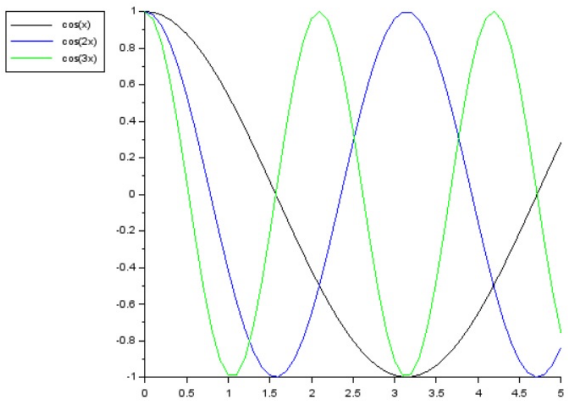
Obrázek 5.6: Graf funkcí cos(x), cos(2x) a cos(3x)

Jak již bylo zmíněno, legendu můžeme vypsat pomocí **leg** (vypíše legendu pouze pod graf) a nebo pomocí funkce **legend**. Syntaxe funkce je legend([h,] string1, string2 , ... [,pos] [,boxed]), kde string1, string2 jsou názvy jednotlivých křivek, které chceme zapsat do legendy. Parametr pos nám určuje, kam legendu umístit. Zapisuje se jako vektor [x,y] a umožňuje vkládání některé z následujících předdefinovaných proměnných.

Tabulka 5.3: Význam legendy

Číslo	Význam
-6.	vykresleno pod dolním levým rohem
-5.	vykresleno nad horním levým rohem
-4.	vykreslena vpravo od pravého dolního rohu
-3.	vykreslena vlevo od levého dolního rohu
-2.	vykreslena vlevo od levého horního rohu
-1.	vykresleno vpravo od pravého horního rohu
1.	default nastavení - vykresleno v horním pravém rohu
2.	vykresleno do horního levého rohu
3.	vykresleno do dolního levého rohu
4.	vykresleno do dolního pravého rohu
5.	vykresleno na místo po kliknutím myší

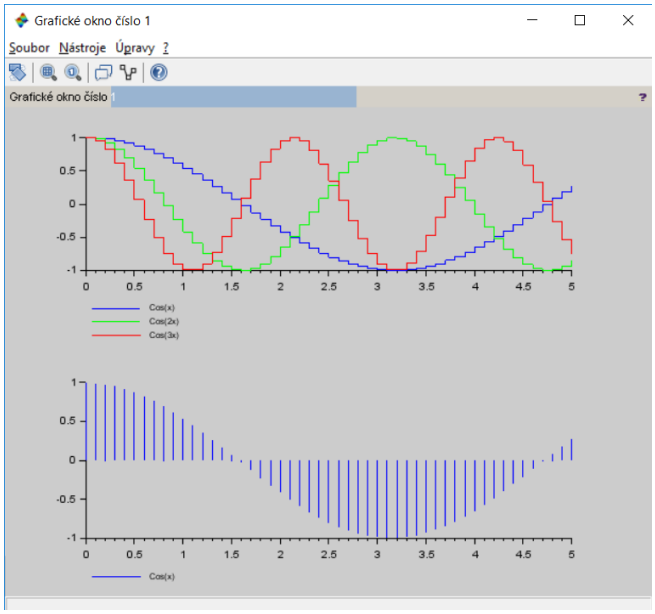
```
x=[0:0.1:5]';
plot2d(x,[cos(x) cos(2*x) cos(3*x)])
legend('cos(x)','cos(2x)','cos(3x)',-2)
```



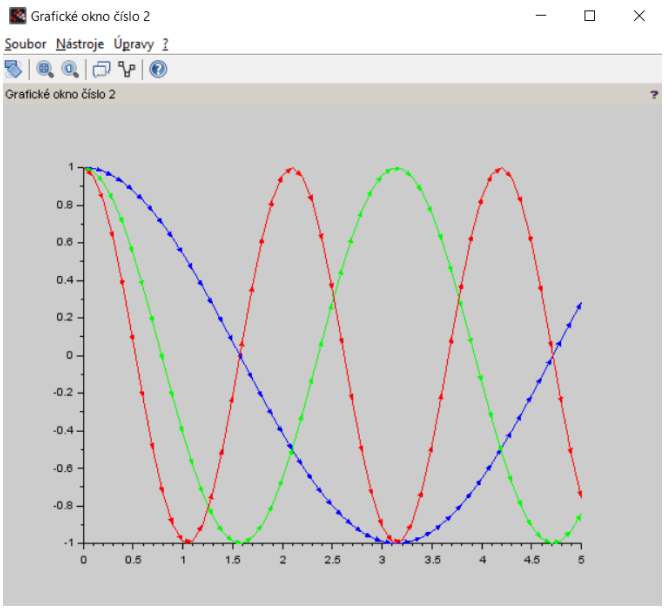
Obrázek 5.7: Vykreslení grafu pomocí funkce legend

Dále zde máme funkce, které vychází z `plot2d` a to: `plot2d2`, `plot2d3` a `plot2d4`, které mají stejnou syntaxi i parametry. Funkce `plot2d2` vykresluje schodkový graf a funkce `plot2d3` vykresluje křivky jako kolmice. Funkce `plot2d4` vykresluje graf pomocí šipek.

```
x=[0:0.1:5]'; figure(1); subplot(2,2,1);
plot2d2(x,[cos(x) cos(2*x) cos(3*x)], leg="Cos(x)@Cos(2x)@Cos(3x)", style = [2,3,5]);
subplot(2,2,2);
plot2d3(x,[cos(x)], leg="Cos(x)", style =[2]);
figure(2);
plot2d4(x,[cos(x) cos(2*x) cos(3*x)],style=[2,3,5]);
```



Obrázek 5.8: Funkce plot2d2, plot2d3



Obrázek 5.9: Funkce plot2d4

V této kapitole jsem čerpala z literatury [4], [8].

6 3D grafy

Scilab podporuje také kromě 2D grafů práci s prostorovou grafikou. Práce s 3D grafy je obdobná práci s 2D grafy, ale musíme myslet na to, že grafy jsou trojrozměrné a jsou vykreslovány do prostorových souřadnic `X, Y, Z`.

Pro vykreslení 3D grafu ve Scilabu existuje funkce `plot3d`. U této funkce lze nastavit mnoho parametrů (barvy, různé způsoby zobrazení grafu, os, legend, apod.). Stejně jako u 2D grafů i z `plot3d` vychází další tři funkce a to `plot3d1`, `plot3d2` a `plot3d3`.

Funkce `plot3d` a `plot3d1` jsou téměř totožné, vyjma toho, že `plot3d1` vykreslí graf v barevné paletě. Funkce `plot3d2` dokáže vykreslit obdélníkové aspekty. Funkce `plot3d3` je opět stejná jako funkce `plot3d2`, nicméně výsledný graf je vykreslen v síťovém modelu.

Funkce `surf` a `mesh` jsou funkce, jež byly přebrány z programu MATLAB pro zlepšení kompatibility a také kvůli zjednodušení práce se Scilabem.

Funkce `param3d` a `param3d1` se používají k vyjádření parametrických grafů. Těmito funkcemi lze vykreslit např. šroubovici.

6.1 Funkce plot3d a plot3d1

Funkce `plot3d(z)` vykreslí plochu `z = f(x,y)` v 3D zobrazení. Souřadnicový systém má tedy tři osy, a to `x, y, z`. Syntaxe příkazu: `plot3d(x,y,z,<opt_rgs>)`, kde `x`, `y` jsou řádkové vektory `n1` a `n2` (souřadnice osy x a y), tyto souřadnice musí být monotónní. `Z` je matice o velikosti (`n1` a `n2`), `z(i,j)` je hodnota plochy v bodě `x(i)` a `y(j)`. Nastavení grafu `<opt_rgs>` obsahuje několik parametrů.

Parametry `theta` a `alpha` jsou hodnoty ve stupních, které udávají pozorovací bod. Parametr `leg` je řetězec definující název každé osy oddělené znakem `@`. Příklad zápisu může vypadat takto: `X@Y@Z`.V základním nastavení nemají osy název.

Parametr `flag` je vektor, který obsahuje tři položky, a to `mode, type, box`. Základní nastavení je definováno jako `flag=[2,8,4]`.

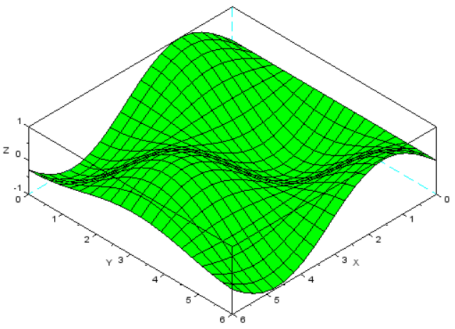
- `mode` vyjadřuje číslo barvy pro vybarvení plochy
  - `mode < 0` vykreslí se plocha v zadané barvě bez mřížky (nastavení barvy lze v tomto případě nastavit i pomocí `color_mode` a `color_flag`)
  - `mode = 0` vykreslí se pouze mřížka nebo síť plochy
  - `mode > 0` vykreslí se plocha příslušné barvy i s mřížkou

- `type` nastavuje měřítko
  - `type = 0` souřadnice jsou zobrazeny podle současného nastavení
  - `type = 1` automatická změna měřítka souřadnic, rozmezí souřadnic vychází z položky `ebox`
  - `type = 2` automatická změna měřítka souřadnic, měřítko vychází z daných dat
  - `type = 3` isometrické nastavení os (všechny tři osy ve stejném měřítku), vychází z položky `ebox`
  - `type = 4` isometrické nastavení os, vychází z daných dat
  - `type = 5` roztažené isometrické nastavení, vychází z položky `ebox`
  - `type = 6` roztažené isometrické nastavení, vychází z daných dat
- `box` nastavuje rámeček kolem grafu
  - `box = 0` žádná položka kolem grafu se nezobrazí (osy, legenda, ani rámeček)
  - `box = 1` neimplementovaný, tedy stejný jako `box = 0`
  - `box = 2` zobrazené pouze osy, které jsou skryté
  - `box = 3` zobrazení os, rámečku a legendy, ale ne stupnice
  - `box = 4` zobrazení všech prvků grafu

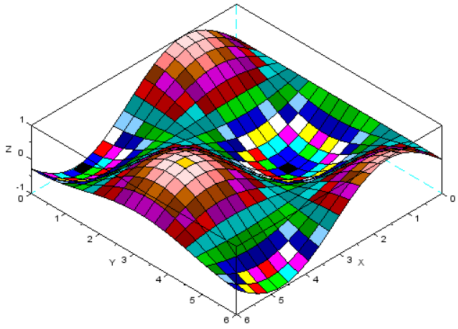
Posledním parametrem je `ebox`, který nastaví hranice vykresleného grafu. Je vyjádřen vektorem `[xmin,xmax,ymin,ymax,zmin,zmax]`. Tento parametr je používán společně s položkou `type` v parametru `flag`. Může nabývat pouze hodnoty 1, 3 nebo 5. Jestliže není zadaný parametr `flag`, `ebox` se ignoruje. V základním nastavení `ebox` není uveden.

Vykreslíme plochu `z = sin(t)×cos(t')` a nastavíme si vhodné pozorovací body `alpha` a `theta`, k tomu si nastavíme parametr `flag`, číslo barvy (3), změnu měřítka souřadnic podle daných dat (2) a nakonec zobrazení všech prvků (4). Pro tyto parametry použijeme funkci `plot3d`. Dále si vykreslíme stejnou plochu, ale pomocí funkce `plot3d1`, kde máme nastavení barevné palety. Vynecháme parametr `flag`. Tím se nám zobrazí základní barevná paleta.

```
t = [0:0.3:2*%pi]';
z~= sin(t) * cos(t');
plot3d(t,t,z,alpha = 50, theta = 45, flag=[3,2,4])
plot3d1(t,t,z,alpha = 50, theta = 45)
```



(a) plot3d

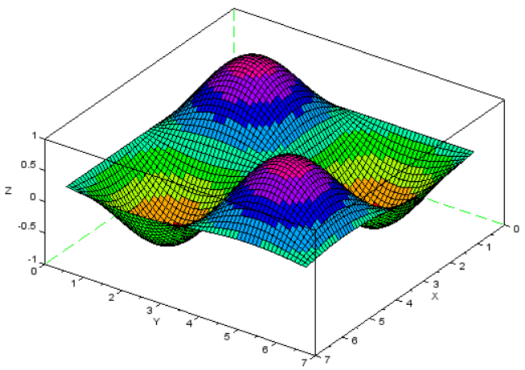


(b) plot3d1

Obrázek 6.1: Ukázka funkce plot3d a plot3d1 s nastavením pozorovacího bodu

Vykreslíme plochu `z = (sin(x))'×(sin(x))` v barevné paletě `hsvcolormap` pomocí funkce `plot3d1`.

```
x = 0:0.1:2*%pi;
z=(sin(x))'*(sin(x));
f = gcf();
f.color_map = hsvcolormap(9);
plot3d1(x, x, z, alpha=70, theta=35);
```



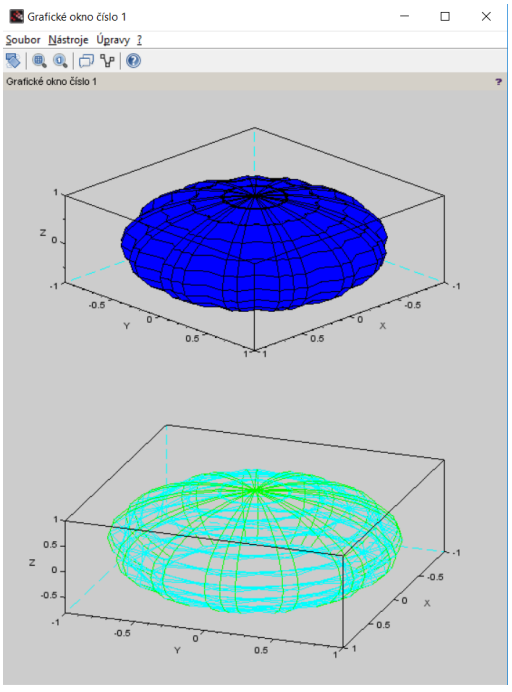
Obrázek 6.2: Graf funkce plot3d1 s barevnou paletou hsvcolormap

6.2 Funkce plot3d2 a plot3d3

Nyní se podíváme na funkce `plot3d2` a `plot3d3`, které se používají pro vykreslení složitějších ploch. Tyto plochy jsou definovány jako `Z = f(X, Y)`, kde `Z` a `X`, `Y` jsou matice popisující danou plochu. Ta je následně poskládaná z obdélníků, což nám umožňuje vykreslit kteroukoliv plochu. Funkce `plot3d3` se od funkce `plot3d2` liší tím, že graf vykreslí v síťovém modulu, nicméně syntaxe zůstává stejná.

Pokud srovnáme funkce `plot3d` a `plot3d2`, tak rozdílem jsou proměnné `x` a `y`, které jsou u `plot3d` monotónní vektory a u `plot3d2` matice. Matice `X`, `Y`, `Z` musí mít stejnou velikost (rozměr). Syntaxe příkazu: `plot3d2(x,y,z,<opt_rgs>)`, kde `<opt_rgs>` obsahuje tytéž parametry `mode, type, box`.

```
u~= linspace(-%pi/4,%pi/2,10);
v~= linspace(0,9*%pi,30);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z~= sin(u)'*ones(v);
figure(1)
subplot(2,1,1)
plot3d2(X,Y,Z, theta = 45, alpha= 45);
subplot(2,1,2)
plot3d3(X,Y,Z, theta = 20, alpha = 45 );
```



Obrázek 6.3: Funkce plo3d2 a plot3d3

6.3 Funkce surf a mesh

Funkce `surf` byla vytvořena pro lepší kompatibilitu s MATLAB syntaxí. Pro zlepšení grafické kompatibility by uživatelé MATLABu měli používat spíše `surf` funkci než `plot3d`. Funkce `surf` vykresluje barevné parametrické povrchy používající čtvercovou mřížku, která je definovaná souřadnicemi `X` a `Y`. Jestliže `X` a `Y` nejsou uvedeny, mřížka je vytvořena s použitím rozměrů matice `Z`. Syntaxe příkazu:

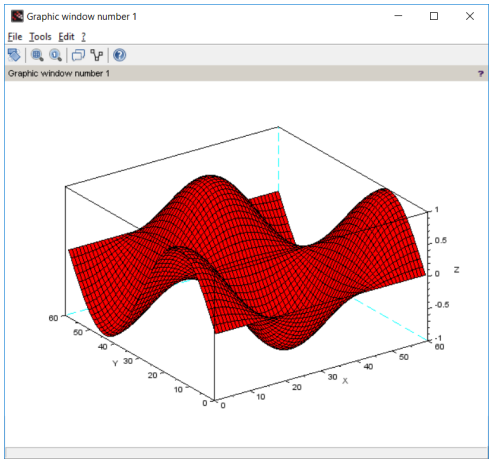
- `surf(Z,<GlobalProperty>)`
- `surf(Z,color,<GlobalProperty>)`
- `surf(X,Y,Z,<color>,<GlobalProperty>)`

`Z` je matice reálných čísel o rozměru `m`×`n`, které definují plochu. Tento argument nesmí být vynechán. Dále parametry `X` a `Y` jsou reálné matice nebo vektory o stejné velikosti. Parametr `color` je volitelný. Je to reálná matice definující barvu pro každé (`X(j)`,`Y(i)`) body na mřížce. Další volitelný parametr je `<GlobalProperty>`, tento argument se vždy zapisuje ve dvojici `PropertyName`,`PropertyValue`. Ty udávají globální nastavení pro všechny nově vytvořené křivky. Používají se u funkce `plot` a `surf`.

Pokud `Z` je jediná specifikovaná matice, pak `surf(Z)` vykreslí matici `Z`. Souřadnice jsou definovány pomocí `X` a `Y`, kde `X` je definováno jako `1:size(Z,2)` a `Y` jako `1:size(Z,1)`. Pokud známe všechny souřadnice `X`, `Y`, `Z`, pak `Z` musí být matice s rozměrem `Z` = [×`n`], kde `X` nebo `Y` je vektor. `X` je vektor o délce `n` a `Y` je vektor o délce `m`. Pokud `X` a `Y` jsou matice, tak její rozměr `X` a `Y` musí být stejný jako rozměr `Z`.

Funkce `mesh` vykreslí graf bez barev jen v síťovém modelu. Má stejnou syntaxi a parametry jako funkce `surf` (vychází z této funkce). Funkce `mesh` má v základním nastavení barvy grafu `color mode = 0` (bílá barva) a `color flag = 0`.

```
u~= linspace(0, %pi*2, 60);
Z~= sin(u)'*cos(u);
figure(1)
surf(Z,'facecol','red')
```

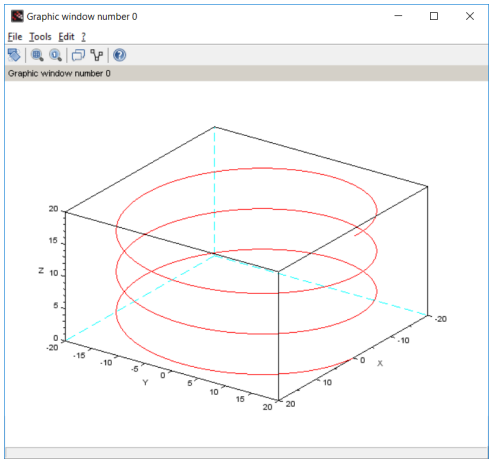


Obrázek 6.4: Funkce surf

6.4 Funkce param3d

Tato funkce se používá pro zobrazení křivek v prostoru. Používá souřadnice `X`, `Y` a `Z`. Dají se nastavit vlastnosti jako rotace, barva, úhly a tloušťka čáry. Pro násobný graf je tu funkce `param3d1`. Syntaxe příkazu: `param3d(x,y,z,[theta,alpha,leg,flag,ebox])`. Jak můžeme vidět, je stejná jako u funkce `plot3d` i s parametry.

```
t = 0:0.01:6*%pi;
param3d(20*sin(t),20*cos(t),t, theta=35, alpha=68);
e = gce();
e.foreground=color('red');
```



Obrázek 6.5: Funkce param3d

V této kapitole jsem čerpala z literatury [4], [8].

7 Práce se soubory

Důležitou součástí funkcí a skriptů je práce se soubory (vstup a výstup dat). Data se ukládají do textových nebo binárních souborů. Většina příkazů vychází z programovacího jazyka C nebo FORTRAN, ale i Scilab má své původní funkce. Scilab poskytuje funkce pro čtení, zápis a také vedlejší funkce. Mnoho vstupních a výstupních funkcí se vyskytuje v párech - jedna je určena pro zápis něčeho, co druhá čte. Do této kategorie spadají také speciální případy, např. psaní a čtení audio souborů. Scilab obsahuje i některé funkce, které jsou ekvivalentní s funkcemi v MATLABu.

7.1 Adresáře

Ve Scilabu, podobně jako v příkazové řádce, můžeme vytvářet složky nebo soubory, které můžeme kopírovat, mazat nebo vypisovat. Uvedeme si několik základních příkladů s příkazy.

```
-->mkdir('C:\Users\Chriss\zkouska');
-->createdir('C:\Users\Chriss\zkouska\test.txt');
-->cd('C:\Users\Chriss\zkouska');
-->pwd
ans  =

C:\Users\Chriss\zkouska

-->dir
ans  =

test.txt\

-->isdir('C:\Users\Chriss\zkouska');
-->createdir('C:\Users\Chriss\zkouska\test2.txt');
-->dir
ans  =

test.txt\ test2.txt\

-->rmdir('test2.txt')
-->dir
ans  =

test.txt\

-->copyfile('C:\Users\Chriss\zkouska\test.txt','C:\Users\Chriss\zkouska2\test.txt');
-->cd('C:\Users\Chriss\zkouska2');
-->dir
ans  =
test.txt\
```

Ve výše uvedeném příkladu jsme nejdříve vytvořili složku pomocí příkazu `mkdir`, ve které jsme odlišným způsobem (`createdir`) vytvořili soubor s příponou `.txt`. Příkazem `cd` jsme se přesunuli do námi vytvořené složky `zkouska`. Poté jsme si vypsali, v jaké složce se nacházíme, použitím příkazu `pwd`. Nechali jsme si vypsat list všech souborů, které se nachází v dané složce příkazem `dir`. Pomocí `isdir` jsme otestovali, zda cesta, kterou jsme zadali, je platná. Ve složce `zkouska` jsme vytvořili další soubor s příponou `.txt`. Opět jsme vypsali list souborů. Dále jsme příkazem `rmdir` vymazali soubor `test2.txt` a poté jsme ověřili, zda je soubor odebraný. V neposlední řadě jsme zkopírovali soubor `test.txt` do složky `zkouska2` pomocí příkazu `copyfile`. Poté jsme se přesunuli do složky `zkouska2`, kde jsme si vypsali list souborů, abychom zjistili, zda se soubor překopíroval.

7.2 Výpis proměnných

Často potřebujeme vypsat hodnoty proměnných či výsledky výpočtů uvnitř funkcí. Explicitně můžeme tyto hodnoty zobrazit pomocí funkce `print` nebo `disp`. Tyto dvě funkce poskytují výpis podobný výpisu z konzole (vizuálně). Ve funkci `print` se první parametr používá pro definici způsobu výstupu. Můžeme použít cestu k souboru zapsanou pomocí řetězce pro zápis do souboru či popis souboru získaný z funkce `file`. Nakonec můžeme také použít předdefinovanou konstantu `%io(2)` pro standardní výstup (výpis do konzole). Formát zobrazení reálných čísel funkcemi `disp` a `print` může být přizpůsoben funkcí `format`.

```
-->a = [1 2 3 4; 4 5 6 7]
a  =

1.    2.    3.    4.
4.    5.    6.    7.

-->print(%io(2),a) // %io(2) - id cislo 2 pro vystup, vystupem je matice a
a  =

1.    2.    3.    4.
4.    5.    6.    7.

-->disp(a) // zobrazeni matice a

1.    2.    3.    4.
4.    5.    6.    7.
```

7.3 Otevíraní a zavírání souboru

Předtím, než budeme číst ze souboru nebo zapisovat do souboru, musí být daný soubor otevřený. Toho dosáhneme pomocí funkcí `fprintMat` a nebo `fscanMat`, které používají název souboru jen pro upřesnění, ze kterého souboru mají číst nebo do kterého mají zapisovat. Tyto funkce vycházejí z programovacího jazyka C. Scilab má své dvě funkce na otevírání souborů, a to `mopen` nebo `file`.

Syntaxe příkazu: `[fd, err] = mopen(file [, mode ])`, kde `file` je cesta k souboru nebo jeho název, `mode` je znak, který specifikuje způsob otevření soubor (čtení nebo zápis). Tato funkce vrací `err`, který nabývá hodnot 0 až -5. Tyto hodnoty představují chyby, které nastaly. Další hodnotou je `fd - file descriptor` (v Matlabu jej nazýváme identifikátor souboru). Jedná se o **identifikátor souboru** (ne o název souboru), jenž se následně používá pro upřesnění toho, ze kterého souboru se má číst nebo, do kterého souboru se má zapisovat.

Ve Scilabu jsou souborové identifikační čísla v rozmezí 1 - 19. ID 1 je používáno pro ukládání historie do souboru `scilab.hist`. ID čísla 5 a 6 (nebo `%io(1)` a `%io(2)`) jsou rezervovaná pro vstup z klávesnice a pro výstup na obrazovku (konzole). Maximální počet souborových ID dostupných pro uživatele je tedy 16, tzn., že v jednu chvíli může být otevřeno 16 uživatelských souborů.

Syntaxe příkazu: `file("open", unit)`, `file("close", unit)`, kde `open` nebo `close` určuje, zda se má soubor otevřít nebo zavřít. Parametr `unit` představuje název nebo cestu k souboru.

Soubory, které jsou otevřeny pomocí funkce `mopen`, musí být zavřeny pomocí funkce `mclose`. Funkce `file` s možností `close` musí být použita pro uzavření souboru, který byl otevřený pomocí funkce `file` s možností `open`.

```
-->print("E:\text.txt",a) //zapis do souboru
-->otevrit = mopen('E:\text.txt','rt'); //otevrit soubor v~modu cteni
-->precist = mgetl(otevrit) //precteni souboru, zda jsme neco zapsali
precist =

! a  =                    !
!                         !
!                         !
!                         !
!   1.   2.   3.   4.     !
!                         !
!   4.   5.   6.   7.     !

-->mclose(otevrit) //zavreni souboru
ans  =

0.
```

7.4 Čtení ze souboru a zápis do souboru

Pro čtení ze souboru a zápis do souboru máme k dispozici několik funkcí. Existují funkce pro textové, binární a zvukové soubory. Pokud máme soubor otevřený, můžeme do něj zapisovat nebo z něj číst.

7.4.1 Funkce mputl a mgetl

Funkce `mputl` zapisuje vektor řetězců do ASCII souboru ve formě sekvence řádků a `mgetl` může přečíst jeden či více těchto řádků.

Syntaxe příkazu: `r = mputl(txt, file_desc)`, kde `txt` je vektor řetězců a `file_desc` je soubor, do kterého se vektor řetezců zapíše. Tato funkce vrací `true`, pokud byla data správně zapsána, v opačném případě vrací `false`.

Syntaxe příkazu: `txt = mgetl(file_desc [,m])`, kde `file_desc` je cesta k souboru nebo `file descriptor`, který vrací funkce `mopen`. Tato funkce vrací sloupcový vektor řetězce.

```
-->ukazka = ['Radek 1'; 'Radek 2'; 'Radek 3']
ukazka =

!Radek 1 !
!        !
!Radek 2 !
!        !
!Radek 3 !

-->mputl(ukazka,'E:\mgetlmputl.txt')
ans  =

T

-->zobraz_vse = mgetl('E:\mgetlmputl.txt')
zobraz_vse =

!Radek 1 !
!        !
!Radek 2 !
!        !
!Radek 3 !
```

Pouze s jedním argumentem čte funkce `mgetl` celý soubor. Druhým argumentem (`m`) můžeme poté blíže specifikovat konkrétní počet řádků, které chceme přečíst.

```
-->dva_radky = mgetl('E:\mgetlmputl.txt',2)
dva_radky =

!Radek 1 !
!        !
!Radek 2 !
```

Pokud požadujeme přečtení více řádků, než se nachází v souboru, funkce skončí s chybovou zprávou. Pokud je druhý argument rovný -1 (výchozí hodnota), funkce čte všechny řádky souboru.

7.4.2 Funkce write a read

Funkce `write` a `read`, podobně jako `mputl` a `mgetl`, se také starají o zápis a čtení souborů. Přestože funkce `write` vyžaduje pouze jméno souboru a data, funkce `read` navíc vyžaduje velikost pole ke čtení ve formátu FORTRAN syntaxe.

Stejně jako `mputl`, funkce `write` zapisuje pole řetězců jednoho sloupce na řádek. Syntaxe příkazu `write`: `write(file-desc,a)`, kde `file-desc` představuje cestu k souboru nebo `file descriptor` získaný z funkce `open`. Parametr `a` je sloupcový vektor řetězců, které chceme zapisovat.

Syntaxe příkazu `read`: `[x] = read(file-desc,m,n,[format])`, kde `file-desc` znovu představuje cestu k souboru nebo `file descriptor`. Parametry `m` a `n` představují dimenze matice řádků a sloupců, které chceme přečíst. Pokud neznáme počet řádků čteného souboru, můžeme nastavit hodnotu `m` na -1 a Scilab přečte celý soubor. Parametr `format` udává způsob čtení souboru ve formátu FORTRAN syntaxe.

```
-->text = ["Delsi retezec pro princip testovani cteni a zapisu souboru"]
text  =

Delsi retezec pro princip testovani cteni a zapisu souboru

-->otevri_soubor = file('open','E:\soubor.txt','unknown')
otevri_soubor =

1.

-->write(otevri_soubor,text)

-->file('close', otevri_soubor)

-->otevri_soubor = file('open','E:\soubor.txt','old')
otevri_soubor =

1.

-->precti_soubor = read(otevri_soubor,-1,1,'(a)')
precti_soubor =

Delsi retezec pro princip testovani cteni a zapisu souboru

-->file('close', otevri_soubor)
```

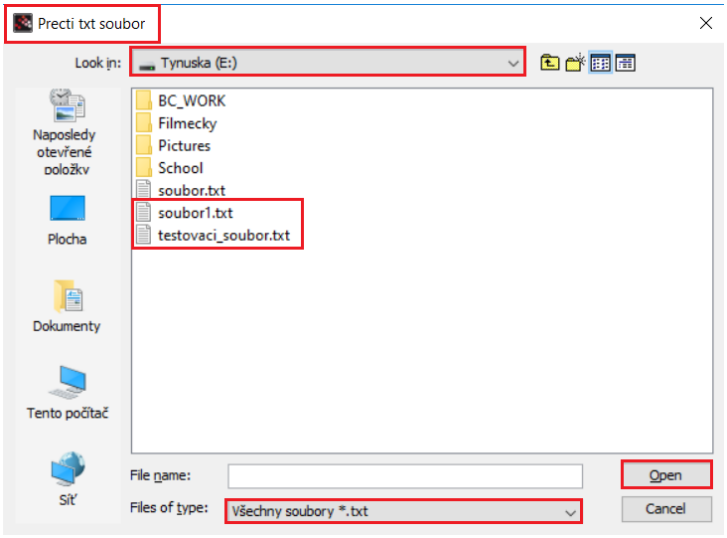
Pokud soubor otevřeme pomocí funkce `file`, je nutné jej znovu uzavřít po provedení operací čtení nebo zápisu. Nejprve si nadefinujeme proměnnou `text`, do které uložíme nějaké řetězec. Následně otevřeme soubor pomocí funkce `file`, která vrátí identifikační číslo souboru. To uložíme do proměnné `otevri_soubor`. Poté do tohoto souboru zapíšeme a následně jej zavřeme. Dále soubor musíme znovu otevřít, nicméně tentokrát se statusem `old`, který říká, že soubor již existuje. Tento otevřený soubor přečteme funkcí `read` a nakonec jej znovu uzavřeme.

7.4.3 Funkce uigetfile a uiputfile

Funkce `uigetfile` zobrazí dialog pro interaktivní výběr souboru. Syntaxe příkazu: `[FileName[,PathName[,FilterIndex]]] = uigetfile([file_mask[,dir[,boxTitle[, multipleSelection]]]])`. Pokud použijeme všechny výstupní parametry, funkce vrací `FileName`, což je matice řetězců obsahující názvy vybraných souborů. Dále vrací `PathName`, který obsahuje cestu k vybraným souborům, a jako poslední parametr vrací `FilterIndex`, který obsahuje index vybraného typu filtru zobrazených souborů. Jestliže použijeme pouze jeden výtupní parametr, funkce vrací plnou cestu k vybraným souborům. Pokud uživatel zruší výběr tlačítkem `Cancel`, výstupní parametry `FileName` a `PathName` budou obsahovat prázdný řetězec a `FilterIndex` hodnotu 0.

Vstupní parametr `file_mask` specifikuje způsob filtrování souborů, `dir` udává výchozí adresář, `boxTitle` definuje název okna a `multipleSelection` povoluje více souborů, pokud je jeho hodnota `true`.

```
-->otevri = file('open','E:\testovaci_soubor.txt','unknown')
otevri =

1.

-->file('close',otevri)

-->[filename, pathname] = uigetfile(filemask='*.txt',dir='E:\', title='Precti txt soubor', %t)
pathname =

E:\
filename =

!soubor.txt testovaci_soubor.txt !
```



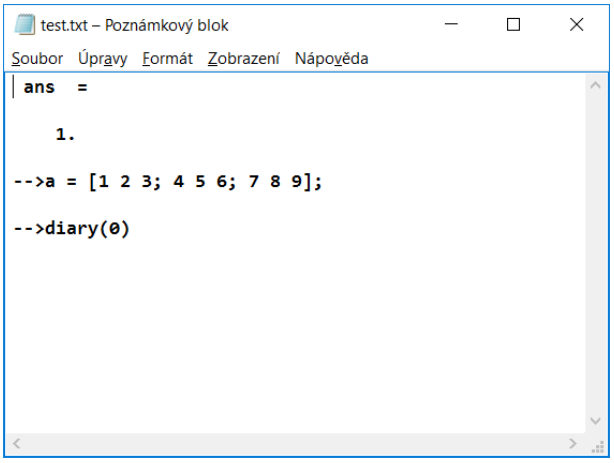
Obrázek 7.1: Otevření dialogu pomocí funkce uigetfile

Funkce `uiputfile` otevře dialog pro uložení souboru. Syntaxe příkazu: `[FileName [,PathName[,FilterIndex]]] = uiputfile([file_mask[,dir[,boxTitle]]])`. Syntaxe této funkce je stejná, jako je tomu u funkce `uigetfile`. S tím rozdílem, že tato funkce neobsahuje vstupní parametr `multipleSelection`, který udává možnost výběru více souborů.

7.4.4 Funkce diary

Scilab umožňuje uložit celou posloupnost příkazů jako text. K tomu existuje funkce `diary`. Na začátku zápisu se zavolá funkce `diary(soubor)`, kde `soubor` je cesta k souboru, do kterého chceme zapisovat. Po zavolání této funkce se všechno, co napíšeme do příkazové řádky, zapíše a uloží do souboru. Tento zápis ukončíme příkazem `diary(0)`. Dnes nabízí větší flexibilitu než `diary` v MATLABu, který pouze vytvoří nový soubor, pokud soubor se zvoleným jménem již neexistuje, jinak připojuje výpis na konec existujícího souboru. `Diary` nahrávání je možné vypnout a zapnout.

```
-->diary('E:\test.txt')
ans  =

1.

-->a = [1 2 3; 4 5 6; 7 8 9];

-->diary(0)
```



Obrázek 7.2: Diary

7.4.5 Funkce save a load

Scilab má své vlastní strojově nezávislé binární formátování pro ukládání objektů. Je možné uložit proměnné do binárního souboru z aktuálního prostředí pomocí funkce `save`. Uložené proměnné mohou být opět načteny pomocí funkce `load`. Výhodou binárních formátů je ukládání numerických dat bez ztráty přesnosti. Pro ukládání grafických dat se používají funkce `xsave` a `xload`.

Syntaxe příkazu: `save(filename [,x1,x2,...,xn])`, kde `filename` je cesta k souboru nebo identifikační číslo souboru. Parametry `x1` až `xn` jsou názvy proměnných, které ukládáme.

Syntaxe příkazu: `load(filename [,x1,...,xn])`, kde máme opět `filename`, který udává cestu k souboru nebo identifikační číslo souboru. Parametry `x1` až `xn` jsou názvy načítaných proměnných.

Funkce `save` a `load` vykonávají stejnou funkci jako funkce `fread` a `fwrite` v MATLABu. Funkce `save` zapisuje jeden, více, nebo dokonce všechny proměnné do souboru. Vstupy a výstupy jsou vytvořeny pomocí funkce `mopen`, kde první argument je cesta k souboru a druhý argument je způsob zpracování (’`rb`’ nebo ’`wb`’).

```
-->a = 2;

-->b = 6;

-->c = "ahoj";

-->otevri = mopen('E:\soubor.bin','wb')
otevri =

1.

-->save(otevri,a,b,c)
Varovani: Scilab 6 nebude podporovat pouzity format souboru.
Varovani: Viz help('save') pro duvody.

-->mclose(otevri)
ans  =

0.

-->clear a // v prohlizeci promennych se vymaze promenna a

-->clear b // to same se provede pro promennou b

-->load('E:\soubor.bin','a','b') //znovu nactene hodnoty vidime v prohlizeci promennych
```

Všimněte si, že proměnné se nepíšou do uvozovek na rozdíl od MATLABu, kde tomu tak je. Scilab nás také upozorňuje na to, že od verze 6.0 nebude podporovaný formát `.bin`. Další funkce pro práci s binárními soubory jsou zobrazeny v níže uvedené tabulce.

Tabulka 7.1: Funkce pro práci s binárními soubory

Funkce	**Popis**
mget	Čtení dat z binárního souboru
mput	Zápis dat do binárního souboru
mgetstr	Čtení řetězce z binárního souboru
mputstr	Zápis řetězce do binárního souboru

7.4.6 Speciální zvukové soubory

Ve Scilabu máme možnost využití zvukových souborů, nicméně zde existuje podpora pouze dvou formátů: `.wav` a `.au`. Scilab poskytuje funkce pro převod souboru `.wav` na `.au` a naopak. Načtený zvuk si poté můžeme poslechnout nebo vykreslit.

Tabulka 7.2: Funkce pro práci se zvukovými soubory

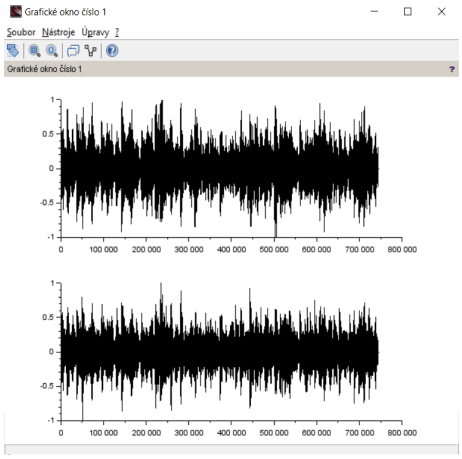
Funkce	**Popis**
auread	Čteni .au audio souborů z disku
wavread	Čtení zvukového souboru .wav
loadwave	Čtení zvukového souboru s příponou .wav
auwrite	Zápis auido souboru na disk s příponou .au
wavwrite	Zápis .wav audio souboru
savewave	Zápis .wav audio souboru
playsnd	Přehrání audio souboru
sound	Přehrání audio souboru

7.4.7 Funkce loadwave, wavread a playsnd

Funkce `loadwave` nám slouží k načtení `.wav` souboru jako matice v programu Scilab. Syntaxe příkazu: `x = loadwave(filename)`, kde `filename` je jméno souboru (nebo cesta k souboru), který chceme načíst. Parametr `x` je matice jednoho řádku pro jeden kanál.

Funkce `wavread` také slouží pro čtení `.wav` souboru. Syntaxe příkazu: `[y,Fs,bits] = wavread(wavfile)`. Parametr `y` je název proměnné, do které uložíme matici souboru. `Fs` je vzorkovací frekvence (počet vzorků za sekundu), v základním nastavení je hodnota rovna `22050`. Parametr `wavfile` je cesta k souboru, který má být načten. Funkce `playsnd` poskytuje zvukový přehrávač.

```
[zkouska,Fs,bits]=wavread('C:\Users\Chriss\ukazka.wav'); // ziskame si vzorkovaci frekvenci a
    pocet bitu na vzorek
figure(1)
subplot(2,1,1)
plot2d(zkouska(1,:)) // vykreslime si prvni kanal
subplot(2,1,2)
plot2d(zkouska(2,:)) // druhy kanal
playsnd(zkouska,Fs) // prehrajeme soubor se ziskanou vzorkovaci frekvenci
disp(Fs) // zobrazime si vzorkovaci frekvenci
disp(bits) // zobrazime pocet bitu na jeden vzorek
```



Obrázek 7.3: První a druhý kanál souboru ukazka.wav

7.4.8 Funkce wavwrite a sound

Funkce `wavwrite` nám umožňuje zapisovat do `.wav` souboru. Syntaxe příkazu: `wavwrite(y, Fs, wavfile)`, kde `y` je vektor nebo matice. `Fs` je opět vzorkovací frekvence a `wavfile` je cesta k souboru, do kterého chceme zapisovat. Funkce `sound` poskytuje zvukový přehrávač stejně jako `playsnd`.

```
f_vz = 44100;
T_vz = 1/f_vz;
t=0:T_vz:2;
A = sin(2 * %pi * 440 * t);
wavwrite(A,'C:\Users\Chriss\ukazka_sin.wav');
[B,Fs,bits]=wavread('C:\Users\Chriss\ukazka_sin.wav');
sound(B,f_vz)
```

V této kapitole jsem čerpala z literatury [3], [9], [10].

8 Výměna dat mezi systémy MATLAB a Scilab

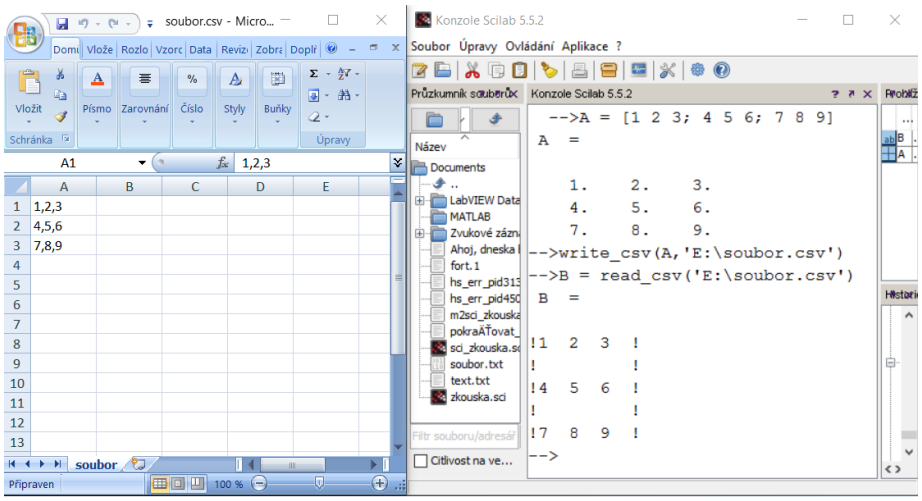
V minulé kapitole jsme se věnovali funkcím pro práci se soubory. Naučili jsme se soubory otevírat, zapisovat, číst a následně je také uzavřít. Ukázali jsme si práci s binárními, textovými a také se zvukovými soubory. V této kapitole se budeme věnovat funkcím pro efektivní výměnu dat mezi systémy MATLAB a Scilab. Oba systémy poskytují funkce pro čtení a ukládání dat do `.csv` a `.xls` souborů. Tyto soubory slouží jako prostředník a nosič dat. Jsou jednoduše editovatelné i v jiných programech.

8.1 Funkce read_csv a write_csv

Tyto funkce slouží pro zápis a čtení jednoduchých dat oddělených oddělovačem (tabulátor, čárka, středník, ...). Syntaxe příkazu: `M = read_csv(fname [,sep])`, kde `fname` je cesta k souboru (řetězec), který chceme číst. Nepovinný parametr `sep` určuje typ oddělovače mezi daty a jeho výchozí hodnota je tabulátor. Funkce vrací matici `M` řetězců načtených hodnot.

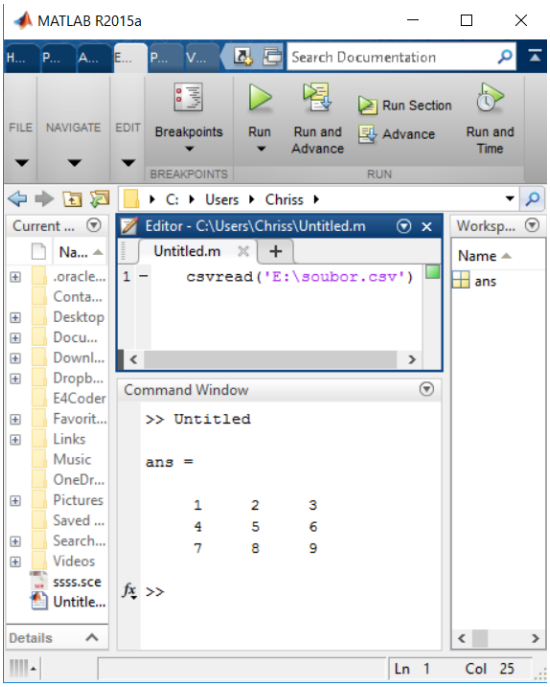
Syntaxe příkazu: `write_csv(M, filename [,sep, dec])`, kde `M` je matice řetězců, které chceme zapisovat. Dále parametr `filename` je cesta k souboru, do kterého chceme zapisovat. Jestliže uvedená cesta k souboru neexistuje, soubor se vytvoří a pokud již existuje, přepíše se. Dalším parametrem je `sep`, který je podobně, jako tomu bylo v předchozím případě, volitelný, určuje typ oddělovače mezi daty a jeho výchozí hodnota je tabulátor. Poslední parametr `dec`, který je opět volitelný, určuje typ desetinné čárky (tečka nebo čárka) a implicitně je roven čárce.

```
A = [1 2 3; 4 5 6; 7 8 9] // definujeme matici, kterou chceme zapsat
write_csv(A,'E:\soubor.csv') // zapiseme matici do souboru
B = read_csv('E:\soubor.csv') // nacteme matici ze souboru do promenne B
```



Obrázek 8.1: Čtení a zápis dat do .csv souboru

Takto zapsaný soubor můžeme poté v MATLABu stejným způsobem přečíst pomocí ekvivalentní funkce `csvread`.



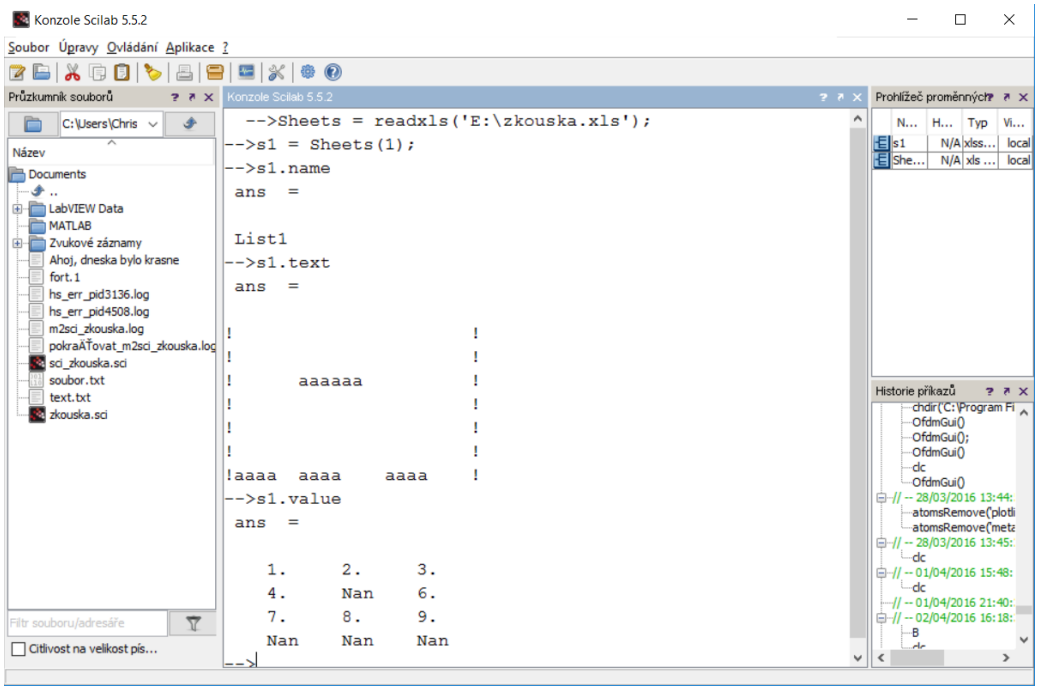
Obrázek 8.2: Načtení .csv souboru v MATLABu

8.2 Funkce readxls, xls_open a xls_read

Další možností je čtení `.xls` souborů, které je podporováno oběma systémy. Scilab však dokáže data pouze **číst** oproti MATLABu, který umí do těchto souborů i zapisovat.

Funkce `readxls` je nejjednodušší z výše zmíněných funkcí. Syntaxe příkazu: `sheets = readxls(file_path)`, kde parametr `file_path` je cesta k souboru. Funkce vrací datovou strukturu `mlist` typu `.xls` s jedním prvkem nazvaným `sheets`. Tento prvek obsahuje seznam listů vytvořených v Excelu, kdy každý list dále obsahuje hodnoty `['name','text', 'value']`, kde `name` je název aktuálního listu, `text` je matice textových hodnot a `value` je matice číselných hodnot.

```
Sheets = readxls('E:\zkouska.xls'); // nacteme .xls soubor
s1 = Sheets(1); // nacteme prvni list excel souboru
s1.name // vypiseme jmeno listu
s1.text // vypiseme matici textovych hodnot
s1.value // vypiseme matici ciselnych hodnot
```



Obrázek 8.3: Načtení .xls souboru

Funkce **xls_open** slouží k otevření `.xls` souborů. Syntaxe příkazu: `[fd, SST, Sheetnames, Sheetpos] = xls_open(file_path)`. Průběh vykonávání této funkce spočívá nejprve v analyzování datové struktury vstupního souboru, ze kterého extrahuje Excel stream. Tento stream je následně uložen do dočasného adresáře, na nějž ukazuje proměnná `fd`, a otevřen. Poté tato funkce čte první list Excel streamu, z něhož extrahuje další informace jako počet listů, jejich jména (`Sheetnames`), adresy listů uvnitř streamu (`Sheetpos`) a hodnotu `STT`, která obsahuje všechny řetězce uvnitř daného listu.

Funkce **xls_read** čte Excel stream dříve otevřený funkcí `xls_open`. Syntaxe příkazu: `[Value,TextInd] = xls_read(fd,Sheetpos)`. Jak je vidět, argument bere hodnoty `fd` a `Sheetpos`, které získáme po otevření daného souboru pomocí funkce `xls_open`. Funkce `xls_read` vrací matici číselných hodnot, obsažených v daném listu prostřednictvím hodnoty `Value`, a matici řetězců, které se taktéž nacházejí v daném listu, prostřednictvím hodnoty `TextInd`.

```
// otevreme .xls soubor z prechoziho prikladu
[fd,SST,Sheetnames,Sheetpos] = xls_open('E:\zkouska.xls')
[Value,TextInd] = xls_read(fd,Sheetpos(1)) // precteme prvni list excel souboru
Value // vypiseme matici ciselnych hodnot
TextInd // vypiseme matici textovych hodnot
mclose(fd) // uzavreme otevreny excel stream
```

8.3 Funkce mfile2sci

Pokud máme skripty napsány v MATLABu, nelze tyto skripty většinou přímo použít ve Scilabu. Tento přechod se ulehčuje poskytnutím nástrojů pro převod kódu MATLABu na jejich ekvivalenty ve Scilabu. Funkce `mfile2sci` patří do sady nástrojů `M2SCI`. Jedná se o funkci, která se stará o převod MATLAB m-souborů do Scilabu. Snaží se vždy, když je to možné, zaměnit volání funkcí MATLABu za jejich ekvivalenty ve Scilabu. Pro jednoduché otevření m-souboru lze zadat pouze `mfile2sci('nazev_m-souboru')`. Tento soubor se nám otevře přímo ve Scilabu, ale nevytvoří `.sci` soubor. Pro vytvoření `.sci` souboru použijeme následující syntaxi: `mfile2sci([M-file-path [,result-path [,recmode [,only-double [,verbose-mode[, prettyprintoutput]]]]]])`

- `m-file-path` je cesta k souboru, který chceme převést
- `result-path` je výsledná cesta adresáře, kde se vytvoří scilabský soubor
- `recmode` je boolean hodnota, která provádí rekurzivní převod (pokud je hodnota true) Musí být nastavená hodnota false v případě jednoho m-file
- `only-double` je boolean hodnota, která předpokládá, že všechny numerické funkce budou použity pouze s numerickými daty (pokud je hodnota true)
- `verbose-mode` nabízí 4 různé úrovně zobrazení informací
  - 0 - nejsou zobrazeny žádné informace
  - 1 - informace jsou zapsány jako komentář ve výsledném .sci souboru
  - 2 - informace jsou zapsány jako komentář ve výsledném .sci souboru a v .log souboru
  - 3 - informace jsou zapsány jako komentář ve výsledném .sci souboru, .log souboru a vypsány do okna Scilabu
- `prettyprintoutput` je boolean hodnota, která formátuje syntaxi výsledného kódu (pokud je hodnota true)

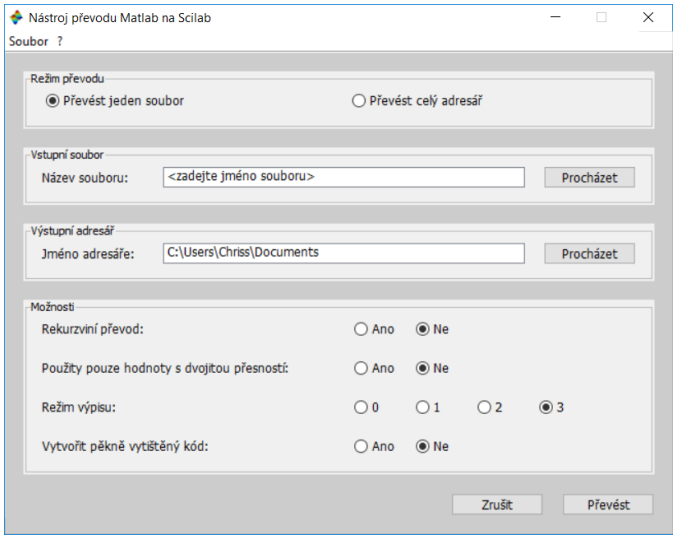
```
-->mfile2sci('E:\zkouska.m','E:\halo\',%f,%f,3,%t)
****** Zacatek relace mfile2sci() ******
Soubor k~prevodu: E:/zkouska.m
Cesta vysledneho souboru: E:/halo/
Rekurzivni rezim: VYP
V~M-souboru jsou pouzity pouze hodnoty s~dvojitou presnosti: NE
Rezim vypisu: 3
Vytvorit formatovany kod: ANO
Cteni M-souboru
Cteni M-souboru Hotovo
Zmena syntaxu...
Zmena syntaxu: Hotovo
Prevod makra na strom...
Prevod M-tree...
Prevod M-tree: Hotovo
Prevod makra na strom: Hotovo
****** Konec relace mfile2sci() ******
ans  =

1.
```

8.4 Převod více m-souborů

Pro převod více souborů existuje ve Scilabu funkce `translatepaths`. Tato funkce konvertuje sadu m-souborů v daném adresáři, kdy každý m-soubor je konvertován pomocí již zmíněné funkce `mfile2sci`. Pokud je funkce volána bez parametru `translatepaths()`, spustí se nástroj Překladač MATLAB na Scilab. Syntaxe příkazu: `translatepaths(dirs_path [,res_path])`, kde `dirs_path` je vektor cest k adresářům obsahující m-soubory a `res_path` cesta adresáře, do kterého se zapisují výsledky převodu.

8.5 Použití nástroje Překladač MATLAB na Scilab

Ukázali jsme si, jak se dá daný soubor nebo sada adresářů převést pomocí funkcí. Další možností je použití nástroje překladač souborů, který převede soubor nebo celý adresář souborů z MATLABu do Scilabu. V horní liště zvolíme tlačítko **Aplikace**, kde klikneme na Překladač Matlab na Scilab. Otevře se okno, ve kterém postupujeme následovně:



Obrázek 8.4: Překladač MATLAB na Scilab

1. Určíme si režim převodu, zda chceme převést jeden soubor či celý adresář.
2. Zvolíme si cestu k souboru nebo adresáři, který chceme převést.
3. Zvolíme místo, kde chceme soubor či adresář uložit.
4. Přizpůsobíme si nastavení rekurzivního převodu, přetypování nenumerických funkcí, režim výstupu a formátování vytištěného kódu.

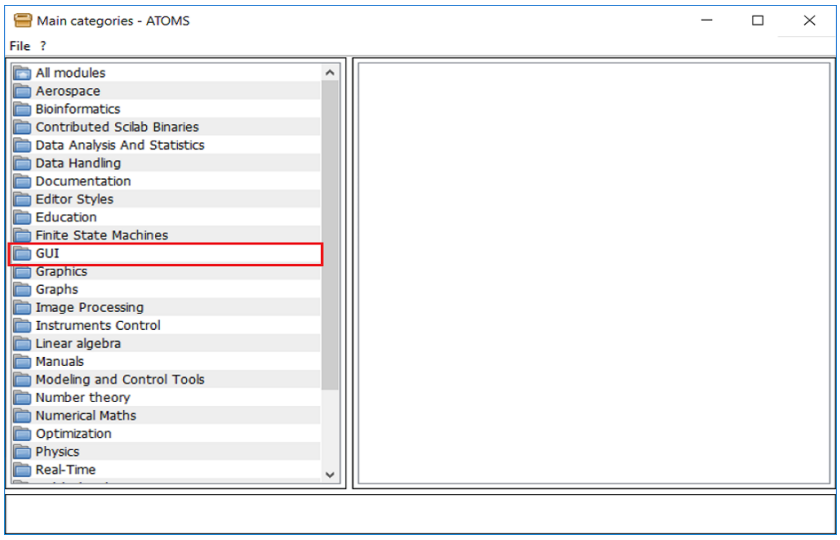
V této kapitole jsem čerpala z literatury [9], [11].

9 Tvorba a konfigurace GUI

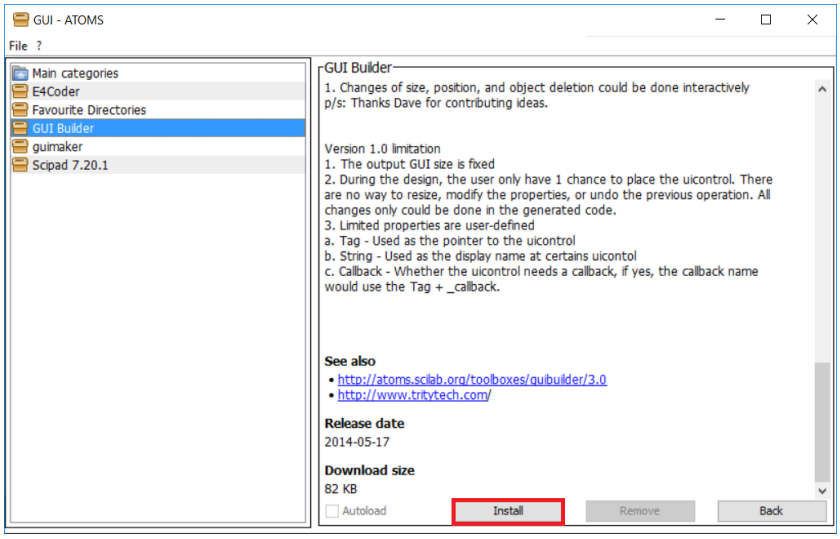
Ve Scilabu je tvorba grafického uživatelského rozhraní možná pouze pokud si nainstalujeme rozšiřující balíček `GUI builder` přes správce ATOMS. Způsob, jak to lze provést a jeho použití, si popíšeme v následujících kapitolách.

9.1 Instalace toolboxu GUI builder

Po otevření programu Scilab označíme myší ikonu **Aplikace** a klikneme na **Správce balíčků - ATOMS**. Následně se nám otevře nové okno, kde si z nabídky **GUI** zvolíme **GUI Builder** a klikneme na tlačítko **Install**.



Obrázek 9.1: Instalace pomocí ATOMS - krok 1



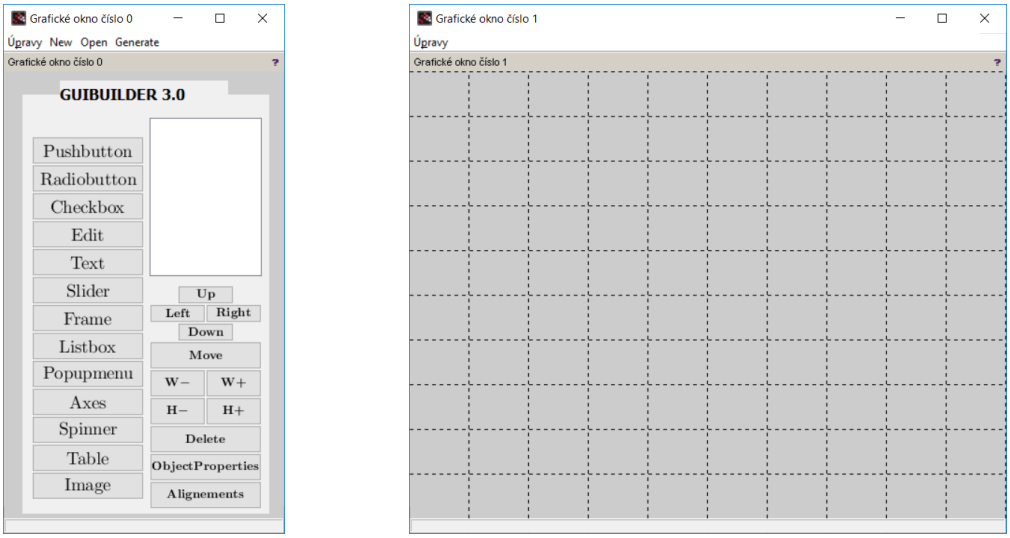
Obrázek 9.2: Instalace pomocí ATOMS - krok 2

9.2 Práce s toolboxem GUI builder

Po skončení instalace program vypneme a znovu zapneme. Zobrazí se zpráva, že **GUI Builder** byl úspěšně nainstalován. Pro jeho spuštění napíšeme do konzole příkaz `guibuilder`.

```
START GUI Builder 3.0 for Scilab 5.5
Load macros
Load help
Type "guibuilder" to start the GUI
```

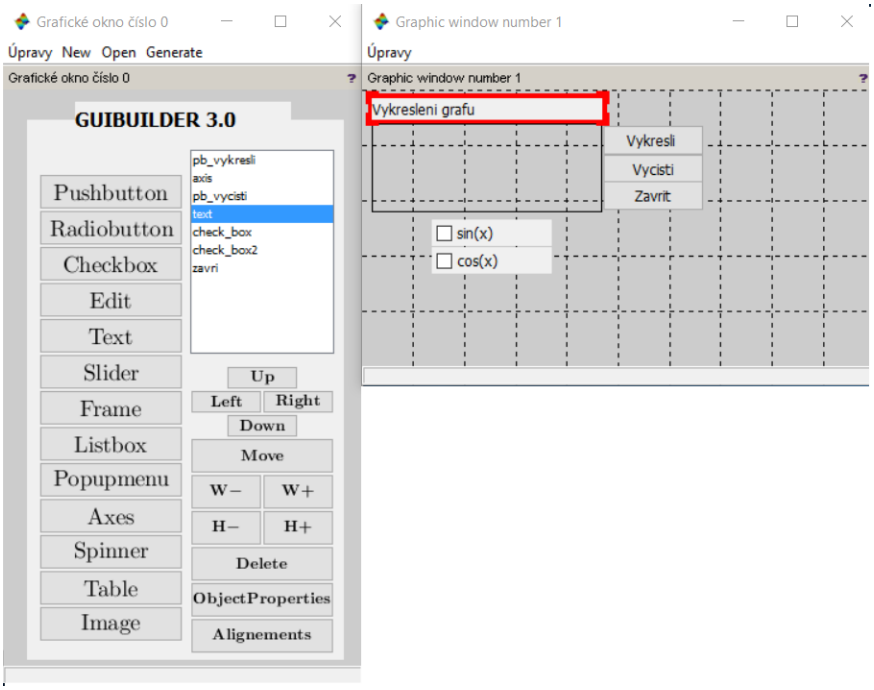
Po spuštění programu se otevřou dvě okna. První okno zobrazuje komponenty, které můžeme použít. Mezi ty patří např. `Pushbutton`, `Radiobutton`, `Checkbox`, apod. Tyto tlačítka lze přetáhnout do druhého okna, které slouží jako kreslící plátno.



(a) Okno s komponentami (b) Kreslící plátno

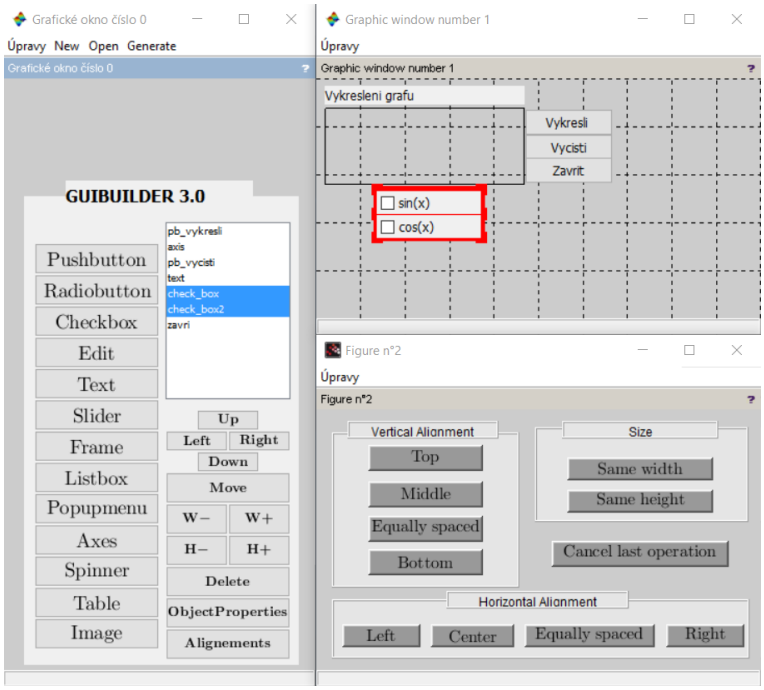
Obrázek 9.3: Úvodní okna pro tvorbu GUI

Po kliknutí na komponentu se nám zobrazí okno, kde musíme zadat **tag** a **String**. Pole **tag** slouží k pojmenování objektu, se kterým pracujeme. Pole **String** slouží k pojmenování komponent, které uvidíme ve výsledném GUI. Po vytvoření grafického uživatelského rozhraní můžeme v okně s komponentami vidět pojmenované objekty. Po kliknutí na objekt v okně s komponentami se na plátně označí příslušná komponenta v červeném rámečku.



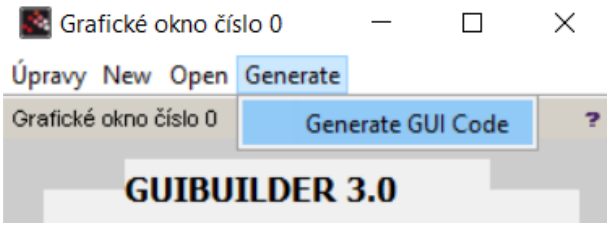
Obrázek 9.4: Označení komponenty

S označenou komponentou můžeme pohybovat všemi směry, měnit její velikost a pozici. Při označení více komponent můžeme pomocí `Alignements` nastavit všem stejnou velikost, poté horizontální nebo vertikální postavení.



Obrázek 9.5: Označení více komponent

Vlastnosti komponenty si můžeme prohlédnout kliknutím na `Object Properties`. Zde se zobrazí jméno objektu, jeho vlastnosti, nastavení, apod. Po vytvoření GUI si musíme nechat vygenerovat kód, se kterým budeme dále pracovat.



Obrázek 9.6: Generování kódu

Vygenerovaný kód si následně uložíme. Po jeho uložení se tento soubor automaticky otevře v editoru SciNotes. Po otevření kódu lze vidět nastavení objektů, které jsme si vytvořili. Nyní musíme naše grafické uživatelské rozhraní naprogramovat. Každý objekt má svou funkci `callback`, do které píšeme samotný kód.

```
function pb_vykresli_callback(handles)
//Write your callback for pb_vykresli here
x = 0:0.1:10;

if  handles.check_box.value == 1 then
plot2d(x,sin(x))
end

if  handles.check_box2.value == 1 then
plot2d(x,cos(x))
end
endfunction
```

Ve výše uvedeném příkladu máme funkci `pb_vykresli_callback(handles)`, jejíž obsah se provede po kliknutí na příslušné tlačítko. V našem případě chceme, aby se vykreslil graf funkce `sin(x)` nebo `cos(x)`. Jestliže bude zaškrtnuté zaškrtávací políčko 1, bude vykreslen graf funkce `sin(x)`. V případě zaškrtávacího políčka 2 se vykreslí graf funkce `cos(x)`. Tato funkce je provázaná s `handles`, který udává vlastnosti daného tlačítka.

```
handles.pb_vykresli=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on',
'FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight',
'normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],
'Min',[0],'Position',[0.47,0.7636585,0.2010444,0.1141732],'Relief','default','SliderStep',
[0.01,0.1],'String','Vykresli','Style','pushbutton','Value',[0],'VerticalAlignment','middle',
'Visible','on','Tag','pb_vykresli','Callback','pb_vykresli_callback(handles)')
```

V této kapitole jsem čerpala z literatury [4].

10 Symbolické výpočty

Zatímco MATLAB nabízí mnoho možností pro práci se symbolickými výpočty, kdy jsme schopni vytvořit takřka jakoukoliv rovnici s využitím symbolických proměnných, které definujeme pomocí `syms`, Scilab nabízí pouze několik předdefinovaných funkcí. Tyto funkce jsou omezeny na základní aritmetické operace, jako je sčítání, odčítání, násobení a dělení (pravé a levé). Dále nabízí funkce pro řešení lineárních kombinací, triangularizací a lineárních systémů.

10.1 Základní aritmetické funkce

Funkce **`addf`** slouží k symbolickému sčítání. Syntaxe příkazu: `c = addf(a,b)`. Tato funkce vrací řetězec znaků `c`, což je `'a + b'`. Dále provádí zjednodušení výrazu, kdy `addf('1','2')` je rovno `'3'` a `addf('0','a')` je rovno `'a'`.

Funkce **`subf`** provádí symbolické odčítání. Syntaxe příkazu: `c = subf(a,b)`. Opět vrací řetězec znaků `'a - b'`, také zjednodušuje výrazy, jako tomu bylo v předchozím případě. Výraz `subf('1','2')` se bude rovnat `'-1'` a `subf('1','a')` se bude rovnat `'-a+1'`.

Funkce **`mulf`** slouží k symbolickému násobení. Syntaxe příkazu: `mulf(d,c)`, Tato funkce vrací řetězec znaků `'c × d'`. Provádí zjednodušení výrazu, v případě `mulf('1','c')` nám vrátí hodnotu `'c'`

Funkce **`ldivf`** a **`rdivf`** provádí levé a pravé dělení symbolických proměnných. Syntaxe příkazů: `ldivf(d,c)` vrací řetězec znaků `'c \ d'`, `rdivf(d,c)` vrací řetězec znaků `'c / d'`. V případě `'c / 1'` a `'1 \ c'` je u obou výrazů výsledek `'c'`.

10.2 Funkce cmb_lin, solve, trisolve trianfml

Funkce **`cmb_lin`** slouží k vyhodnocování lineárních kombinací. Syntaxe příkazu: `[x] = cmb_lin(alfa, x, beta, y)`, kdy vrací výsledek `'alfa×x-beta×y'`.

Funkce **`solve`** a **`trisolve`** slouží k řešení lineárních systémů. Syntaxe příkazů: `[x] = solve(A, b)` a `[x [,sexp]] = trisolve(A,b [,sexp])`. Obě funkce berou jako argumenty `A` a `b`, což jsou matice (respektive vektory) vstupních řetězců. Výsledkem je rovnice `'A × x = b'`, kde `A` je horní trojúhelníková matice, která se skládá z textových řetězců. Funkce `trisolve` navíc poskytuje další argument `sexp`, což je vektor obecných podvýrazů k `A`, `b`, `c`.

Nakonec funkce **`trianfml`** slouží k provádění symbolických triangularizací. Syntaxe této funkce: `[f [,sexp]] = trianfml(f [,sexp])`. Funkce vyhodnocuje symbolické triangularizace matice `f`. Triangularizace se provádí elementárními řádkovými operacemi. Argument `sexp` určuje sadu obecných výrazů ukládaných daným algoritmem.

10.3 Funkce evstr

V případě symbolických výpočtů také často potřebujeme dosadit konkrétní hodnoty za výsledné rovnice. K tomu slouží funkce **evstr**, jejíž syntaxe je následující: `H = evstr(Z)`. Funkce vrací výsledek vyhodnocených řetězců matice `Z`, kde každý řetězec musí být definován jako validní výraz Scilabu. Pokud v průběhu výpočtu nastane chyba, Scilab tuto chybu vypíše do konzole. Poté ji můžeme opravit. Více o tomto problému nalezeme v kapitole 4.4.1.

```
matice = ['a','b','a+b';'0','-b','-a';'0','a-b','-1']; // nadefinujeme matici se symbolickymi
    promennymi
tri = trianfml(matice) // prevedeme pomoci trianfml do horni trojuhelnikove matice
a = 1; // priradime hodnoty pro a i b
b = 2;
num = evstr(tri)//pomoci evstr priradime cisla symb. promennym a vypocita vysledek, vraci
    ciselnou podobu

tri  =

!a  b   a+b       !
!                 !
!0  -b  -a        !
!                 !
!0  0   b+(a-b)*a !

num  =

1.    2.    3.
0.  - 2.  - 1.
0.    0.    1.
```

V této kapitole jsem čerpala z literatury [7].

Závěr

Cílem této bakalářské práce bylo vytvořit podrobný návod pro práci se systémem Scilab jako alternativou komerčního systému MATLAB a Simulink při zpracování signálu. Také by tato práce měla zjednodušit uživatelům MALTABu přechod mezi systémy MATLAB a Scilab. Je na místě poukázat na to, že i software, který je dostupný zdarma, může být velice dobrou náhradou licencovaného systému.

Při práci v programu Scilab je nutné dbát na použití jednotlivých funkcí, jelikož ne všechny funkce v MATLABu mají své ekvivalenty ve Scilabu. Některé funkce zde chybí, jiné se liší ve vstupních a výstupních parametrech, další jsou nahrazeny jinými funkcemi nebo se jinak jmenují. Nevýhodou Scilabu může být málo propracovaná nápověda, která je nepřehledná.

Při zpracování mé bakalářské práce byl největší problém v kapitole Ladění zdrojového kódu, jelikož ve Scilabu chybí grafický debugger. Nastavení breakpointů, které jsou nedílnou součástí ladění zdrojového kódu, je možné pouze v konzoli pomocí specifických funkcí nebo výpisů chybových zpráv. Použití funkcí je uživatelsky méně přívětivější, než je tomu v případě použití nástrojů v grafickém uživatelském rozhraní, a celý proces ladění programu se stává složitější.

Další omezení Scilabu se projevilo v kapitole Práce se symbolickými výpočty. Zatímco v MATLABU jsme schopni vytvořit takřka jakoukoliv rovnici, ve Scilabu tuto možnost nemáme. Jsme omezeni pouze na práci se základními aritmetickými operacemi, kdy zde najdeme funkce pro sčítání, odčítání, násobení a dělení symbolických výrazů a jejich následné zkrácení.

V každé kapitole se plně věnuji dané problematice, kde jsou uvedeny příklady s podrobným popisem. Zároveň srovnávám rozdíly práce mezi systémy MATLAB a Scilab. Aby byl Scilab plnohodnotným konkurentem MATLABu, bylo by potřeba zpracovat velké množství problematiky. Nicméně může sloužit dobře jako vědecká kalkulačka, program pro tvorbu 2D a 3D grafů a také jako programovací nástroj.

Literatura

[1] CAMPBELL, Stephen L., Jean-Philippe Chancelier a Ramine Nikoukhah. Modeling and simulation in Scilab/Scicos with ScicosLab 4.4. New York: Springer, 2010. ISBN 978-1-4419-5526-5.

[2] BAUDIN Michaël, Consortium Scilab – DIGITEO. Introduction to Scilab. Scilab. [online]. 2010 [cit. 2015-12-17]. Dostupné z: http://forge.scilab.org/index.php/p/docintrotoscilab/downloads/get/introscilab_v1.7.pdf

[3] RIETSCH, Eike. An Introduction to Scilab from a Matlab User's Point of View. MARS Lab. [online]. 2001-2002 [cit. 2015-12-19]. Dostupné z: http://mars.uta.edu/mae4310/simulation/Scilab4Matlab2.6-1.0.pdf

[4] Scilab Help. Scilab. [online]. 2011-2015 [cit. 2016-12-19]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/index.html

[5] Console. Scilab. [online]. 2011-2015 [cit. 2016-12-20]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_c9c3cbfdaafb0a222c2a8ed83108bc59.html

[6] Debugging. Scilab. [online]. 2011-2015 [cit. 2016-01-24]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_f30f4937bdb6cfca519a6e632d13c988.html

[7] Symbolic. Scilab. [online]. 2011-2015 [cit. 2016-01-24]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_7b914b0b010be1485badc529f41ecd1f.html

[8] Graphics. Scilab. [online]. 2011-2015 [cit. 2016-01-28]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_fbb832a4db5f8ce05fb634eec24fa41c.html

[9] Files : Input/Output functions. Scilab. [online]. 2011-2015 [cit. 2016-02-05]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_efa4cbf5c15e166f34901f827390e756.html

[10] Sound file handling. Scilab. [online]. 2011-2015 [cit. 2016-02-23]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_d11cd0f9e362390f953e7199c5bb4b3a.html

[11] Spreadsheet. Scilab. [online]. 2011-2015 [cit. 2016-02-23]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_c76aa854271b0bbdb796e15512af62cf.html

[12] Control flow. Scilab. [online]. 2011-2015 [cit. 2016-03-01]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/control_flow.html

[13] Trigonometry. Scilab. [online]. 2011-2015 [cit. 2016-03-11]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_99038107015b1d789de50bf92f154a85.html

[14] Log – exp – power. Scilab. [online]. 2011-2015 [cit. 2016-03-14]. Dostupné z: https://help.scilab.org/docs/5.5.2/en_US/section_04cf1ffa9b1efa819e5ac20bbde28fce.html

A Instalace programu Scilab (Windows) v krocích

Obsah této přílohy je na přiloženém CD

B Tvorba a konfigurace GUI

Obsah této přílohy je na přiloženém CD

C Skripty a funkce

Obsah této přílohy je na přiloženém CD